Tampereen yliopisto

Aleksi Luukko

CMS-JÄRJESTELMIEN SUORITUSKYVYN OPTIMOINTI JA PARANTAMINEN

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Kesäkuu 2019

TIIVISTELMÄ

Aleksi Luukko: CMS-järjestelmien suorituskyvyn optimointi ja parantaminen
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Kesäkuu 2019

Web-pohjaisten CMS-järjestelmien suorituskyvyn merkitys on erittäin suuri. Sivuston suorituskyky vaikuttaa muun muassa käyttökokemukseen ja asiakastyytyväisyyteen sekä syntyvään laatuvaikutelmaan. Joskus suorituskyvyn ongelmien taustalla voi olla myös järjestelmässä piileviä laatuongelmia, kuten runsaasti teknistä velkaa tai rappeutunut kokonaisarkkitehtuuri.

Tässä tutkielmassa tehdään katsaus CMS-järjestelmiin sekä ohjelmistojen laatuun, laadun arviointiin ja metriikoihin. Tutkielman tapaustutkimuksen kohteena on erään IT-alalla toimivan yrityksen kehittämä ja markkinoima web CMS -järjestelmä. Suorituskyky on muodostunut tässä järjestelmässä ongelmalliseksi. Tutkielmassa pyritään löytämään keinoja, joilla web CMS -järjestelmien suorituskykyä voidaan parantaa ja optimoida yleisellä tasolla. Löydettyjä keinoja pyritään soveltamaan tapaustutkimuksen kohdejärjestelmään sen suorituskyvyn parantamiseksi ja optimoimiseksi.

Tapaustutkimuksen kohteena olevan web CMS -järjestelmän suorituskykyä onnistuttiin parantamaan ja optimoimaan löydetyillä keinoilla merkittävästi. Erityisesti keskeisiksi keinoiksi nousivat datan käytön parantaminen, toiston purkaminen, kapselointi, abstraktointi ja keskeisten funktioiden parantaminen. Tutkielman yhteydessä tehtiin myös havainto tapaustutkimuksen kohteena olevan web CMS -järjestelmän laatuongelmista ja annetaan ehdotuksia laatutason parantamiseksi myös jatkossa.

Avainsanat: Web CMS -järjestelmä, suorituskyky, tekninen velka, kokonaisarkkitehtuurin rappeutuminen

1 Johdanto

Sisällönhallintajärjestelmien (CMS) suorituskyky on moniulotteinen asia. Suorituskyvyllä voidaan viitata esimerkiksi järjestelmän kykyyn vastata suureen yhtäaikaiseen kysyntään tai toisaalta siihen, kuinka nopeasti ja tehokkaasti se pystyy suorittamaan yksittäisiä siltä vaadittuja tehtäviä. Tässä tutkielmassa käsitellään suorituskykyä pääasiassa *web-pohjaisen CMS-järjestelmän* (Web CMS) julkisen puolen näkökulmasta.

Suorituskykyinen web CMS -järjestelmä tarjoaa paremman käyttökokemuksen ja vähentää käyttäjän tekemiä virheitä järjestelmän nopeiden vastausten avulla. Nykyään verkkosivuilla jo kahdenkin sekunnin vastausaika käyttäjän suorittamaan toimintoon saatetaan nähdä hyvin haitallisena ja toistuessaan jopa erittäin ärsyttävänä kilpailuhaittatekijänä järjestelmän käytössä. Käyttäjät olettavatkin nykyään järjestelmien olevan käytettävissä heti ja vastaavan tehtyihin kutsuihin käytännössä välittömästi [Laird ja Brennan 2006]. Tämä näkyy myös siinä, että esimerkiksi verkkokauppiaana menestymiseen on liitetty myös olennaisena osana käytössä olevan verkkokauppa-alustan suorituskyky ja saavutettavuus [Chosin ja Ghaffari 2017].

Vaikka suorituskyky saatetaankin mieltää lähinnä yllä kuvatun kaltaiseksi käyttökokemukseen liittyväksi asiaksi, on sen merkitys kuitenkin tätä laajempi. Pitkäikäisen tietojärjestelmän arkkitehtuuri rappeutuu vähitellen samalla tapaa kuin eroosiota tapahtuu myös luonnossa [Alcocer ja Bergel 2015]. Mikäli uusien toimintojen ja rakenteiden annetaan vähitellen kasautua vanhojen päälle ilman huolenpitoa järjestelmän kokonaisarkkitehtuurista, voi lopputulemana olla myös suorituskyvyn lasku jokaisen kutsun yhteydessä suoritettavan käsittelyn määrän kasvaessa. Riskitekijöinä koodipohjan eroosiolle voidaan pitää muun muassa järjestelmän ikää, kokoa, kompleksisuutta, suurta ja kasvavaa vaatimusten joukkoa sekä kehitysorganisaatiosta johtuvaa painetta [Eick *et al.* 2001]. Koodipohjan eroosion myötä on joissain tapauksissa tyypillistä, että virhealttius järjestelmän sisällä kasvaa, jolloin heikko suorituskyky voi myös kieliä piilevästä ja perustavanlaatuisesta laatuongelmasta [Eick *et al.* 2001].

Tämä tutkielma on tapaustutkimus, jossa tehdään katsaus CMS-järjestelmiä ja ohjelmistojen laatua käsittelevään kirjallisuuteen sekä käsitellään tapaustutkimuksen kohteena erään IT-alalla toimivan ohjelmisto- ja palveluratkaisuja tuottavan ja tarjoavan yrityksen web CMS -järjestelmän ja verkkokaupan yhdistelmän suorituskyvyn nykytilaa ja pyritään parantamaan sekä optimoimaan sitä kirjallisuudesta löydettyjen ratkaisujen avulla. Suorituskyvyn osalta keskitytään erityisesti järjestelmän vastausaikojen pienentämiseen järjestelmän suorituskyvyssä ongelmalliseksi muodostuneella PHP-pohjaisella backend-puolella järjestelmän keskeisimpien sivujen osalta. Löydetyissä ratkaisuissa pyritään myös huomioimaan järjestelmän kokonaisarkkitehtuurin kehitys ja tukemaan sen kehitystä.

Tutkielman tavoitteena on löytää keinoja, joilla web CMS -järjestelmän suorituskykyä voidaan optimoida ja parantaa olemassa olevien ja yleisesti käytettyjen mittaristojen ja ratkaisujen avulla. Lisäksi pyritään toteuttamaan näitä ratkaisuja tapaustutkimuksen kohteena olevaan web CMS -järjestelmään ja analysoimaan, kuinka paljon sen suorituskykyä voidaan todellisuudessa nostaa löydettyjen ratkaisujen avulla. Tutkimuskysymyksinä onkin siis, miten web CMS -järjestelmien suorituskykyä voidaan optimoida ja parantaa olemassa olevien ja yleisesti käytettyjen mittaristojen ja ratkaisujen avulla sekä miten näitä ratkaisuja ja mittaristoja voidaan hyödyntää tapaustutkimuksen kohteena olevassa web CMS -järjestelmässä. Tutkielma on siis tapaustutkimus, jossa pyritään ymmärtämään tapaustutkimuksen kohteena olevan web CMS -järjestelmän piirteitä ja soveltamaan siihen olemassa olevia ratkaisuja [Eisenhardt 1989].

Tapaustutkimus on tutkimustapa, jonka avulla pyritään tarkastelemaan ja ymmärtämään jossakin tietyssä tapauksessa ilmeneviä tutkittavia ilmiöitä. Tapaustutkimuksessa voidaan tavoitella erilaisia lopputulemia, kuten tämän tutkielman tapauksessa teorian testaamista jossakin yhteydessä tai teorian luomista jossain tapauksessa käytettyjen sovellutusten perusteella [Eisenhardt 1989]. Tämän tutkielman tapauksessa tapaustutkimuksen osalta sovelletaan menetelmää, jossa tunnistettua ongelmaa pyritään ratkaisemaan olemassa olevien erilaisten keinojen avulla iteratiivisen prosessin avulla. Tällöin jokaisen iteraation kohdalla voidaan arvioida saavutettuja tuloksia ja muutoksia. [Eisenhardt 1989, Woodside 2010]

Tutkielman alussa luvussa 2 käsitellään erilaisia CMS-järjestelmiä yleisellä tasolla ja tutustutaan tarkemmin web CMS -järjestelmiin sekä niiden ominaispiirteisiin. Lisäksi paneudutaan dynaamisiin web CMS -järjestelmiin ja niissä piileviin haasteisiin suorituskyvyn näkökulmasta. Luvussa tarkastellaan myös yleisesti web-pohjaisissa järjestelmissä käytetyn PHP-kielen aiheuttamia ja yleisesti tunnistettuja suorituskyvyn ongelmia ja rajoitteita. Luvussa 3 siirrytään käsittelemään ohjelmistojen laatua, laadun arviointia ja erilaisia mittauksessa käytettäviä metriikoita. Siinä tarkastellaan minkälaista osaa laatu näyttelee osana ohjelmiston elinkaarta ja miten laatu sekä laadun ylläpito ja parantaminen olisi hyvä ottaa ohjelmiston elinkaaressa huomioon. Luvussa tarkastellaan myös erilaisia laadun mittaukseen ja arviointiin liittyviä keinoja sekä otetaan lyhyt katsaus olemassa oleviin laatujärjestelmiin. Lopuksi paneudutaan vielä laadun parantamisessa käytettyihin erilaisiin keinoihin sekä käytäntöihin, joilla laatua pystytään vähitellen nostamaan ylöspäin.

Tutkielman luvussa 4 painotus vaihtuu kohti tapaustutkimuksen kohteena olevan web CMS -järjestelmän analysointia. Luvun alussa kuvataan järjestelmä ensin pääpiirteittäin, minkä jälkeen esitellään järjestelmän suorituskyvyn mittaamista varten tämän tutkielman yhteydessä kehitetyt mittaristot. Luvussa kuvataan myös suorituskyvyn mittauksessa käytetty testiympäristö sekä mittaukseen valikoituneet testitapaukset.

Luvussa 5 analysoidaan tapaustutkimuksen kohteena olevan web CMS -järjestelmän suorituskyvyn ongelmia mittaristoilla tutkielman yhteydessä suoritettavien testien avulla. Käsiteltävät ongelmat jakautuvat pääasiassa epätehokkaaseen datan ja tietokannan käsittelyyn, epätehokkaaseen objektien ja funktiokutsujen käyttöön, vanhentuneisiin ja hankalasti ylläpidettäviin tietorakenteisiin sekä yleisellä tasolla järjestelmän kokonaisarkkitehtuurin tilanteeseen.

Tutkielman luvussa 6 käsitellään tapaustutkimuksen kohteena olevaan web CMS -järjestelmään tehtäviä muutoksia, joilla suorituskyvyn tasoa pyritään nostamaan. Suorituskyvyn parantamiseen sovelletut ratkaisut koostuvat muun muassa datan käytön parantamisesta ohjelmakoodin kapseloinnin ja abstraktoinnin avulla, välimuistin käytön lisäämisestä valikoitujen tietokantakutsujen ja objektien luonnin yhteyteen sekä epätehokkaiden funktioiden ja tietorakenteiden refaktoroinnista tehokkaammiksi. Tutkielman lopussa esitellään tapaustutkimuksen kohteena olevaan web CMS -järjestelmään tehtyjen parannusten merkittäviä vaikutuksia järjestelmän suorituskykyyn sekä pohditaan minkälaisin keinoin suorituskykyä ja laatua voitaisiin edelleenkin parantaa ja ottaa monipuolisemmin huomioon järjestelmän tuotekehityksessä.

2 CMS-järjestelmät

CMS-järjestelmät ovat varsin laaja käsite, ja niillä voidaan tarkoittaa kaikkia sellaisia tietojärjestelmiä, joilla sisältöä hallitaan, luodaan ja viedään ihmisten kulutettavaksi suunnitellusti [Boiko 2001]. Tähän mahtuu mukaan hyvin paljon eri tarkoituksia varten räätälöityjä järjestelmiä, kuten esimerkiksi digitaalisten sisältöjen hallintaa, yritysten ja organisaatioiden tietosisältöjen hallintaa sekä web-sisältöjen hallintaa [Barker 2016].

CMS-järjestelmien moninaista luonnetta ymmärtääkseen on hyvä määritellä ensin, mitä sisältö itsessään tarkoittaa. Yleisellä tasolla sisällöksi voidaan määritellä sellainen tieto, joka tuotetaan jonkinlaisen editoriaalisen prosessin läpi ja joka on loppujen lopuksi tarkoitettu jollain tasolla ihmisten saataville ja käytettäväksi [Barker 2016]. Sisältö ei siis ole sama asia kuin tietokoneen tai jonkin tietojärjestelmän käsittelemä data tai muu tieto. Sisältö on aina datasta tai tiedosta jalostettua informaatiota sellaisessa muodossa, jossa se voidaan säilyttää ilman tietosisällön merkittävää laskua [Boiko 2005]. Tällainen sisältö voi koostua esimerkiksi kuvista, tekstistä, lainauksista tai videosta.

Sisällön määritelmän myötä on erittäin tärkeää huomata, ettei CMS-järjestelmä tuota itsessään tietosisältöä; CMS-järjestelmän tarkoitus on antaa työkalut sisällöksi muotoillun tiedon hallintaan [Boiko 2005]. CMS-järjestelmän avulla käyttäjä voi yleensä itse tehdä valinnan siitä, minkälaisella ajanjaksolla sekä minkälaisin keinoin ja tavoin sisältöä tuodaan sitä kuluttavien tahojen saataville rajatusti tai rajoittamatta [Boiko 2001]. CMS-järjestelmän tehtävä onkin pitää huoli siitä, missä kunnossa sisältö on, kuka sitä voi käyttää ja miten sisältö liittyy muuhun sisältöön [Barker 2016].

Vaikka tänä päivänä CMS-järjestelmiä käsiteltäessä ajatellaan helposti kyseessä olevan täysin digitaalinen käsite, ovat sisällönhallinnan juuret kuitenkin paljon syvemmällä analogisessa maailmassa. Esimerkiksi kirjasto on tietynlainen CMS-järjestelmä, jossa tietosisällön muodostavat kirjastossa tarjolla olevat julkaisut ja hallinnan taas kirjaston erilaiset prosessit, jotka voivat olla myös digitaalisia. Sisältöjen hallinnan voidaankin ajatella syntyneen ilmiönä heti, kun sisältöäkin on alettu tuottamaan ja hallitsemaan jossakin jäsennellyssä muodossa [Barker 2016]. Tämän tutkielman kontekstissa tulemme kuitenkin jatkossa keskittymään digitaalisiin CMS-järjestelmiin näistä puhuttaessa.

Digitaaliset CMS-järjestelmät mahdollistavat huomattavasti monipuolisemman tavan hallita sisältöä kuin analogiset järjestelmät. Digitaalisessa formaatissa tietoa ja dataa voidaan muokata helpommin erilaisiksi tietosisällöiksi, jolloin CMS-järjestelmän erilaisten käyttökohteiden ja sovellutusten määrä kasvaa. Tarvittaessa digitaalisen CMS-järjestelmän avulla pystytään myös tuottamaan sisältöä analogiseen muotoon esimerkiksi tulosteiden välityksellä ja muokkaamaan sisällön muotoa uusia erilaisia julkaisuja varten. Onkin tyypillistä, että nykyaikaiset digitaaliset CMS-järjestelmät ovat

toiminnallisuudeltaan varsin laajoja ja että toiminnallisuuden määrä vaihtelee hyvin paljon järjestelmän käyttökohteen mukaan [Boiko 2001].

Riippumatta siitä, minkälaisesta ja kuinka laajasta CMS-järjestelmästä on kyse, sisältävät ne yleensä kuitenkin jonkinlaisia toteutuksia kaikille CMS-järjestelmille tyypillisistä toiminnallisuuden konsepteista. Näitä yleisiä konsepteja on tunnistettu kolme: sisällön keruu johonkin tietovarantoon, kerätyn sisällön hallinta jollain tasolla sekä tämän sisällön julkaisun hallinta jollain tasolla. [Boiko 2005] Nämä kolme konseptia voivat olla laajemmissa järjestelmissä täysin omina itsenäisinä toimintojen kokonaisuuksinaan. Vastaavasti taas pienemmissä järjestelmissä konseptit voivat olla toteutettuina toisiinsa liitettyinä toiminnallisuudeltaan hyvin rajattuina osioina [Boiko 2005].

CMS-järjestelmien kolmesta konseptista ensimmäinen on sisällön keruu johonkin tietovarantoon. Yksinkertaisimmillaan tämä voi olla esimerkiksi tekstin tallentamista tietokantaan ilman sen kummempaa tietosisällön käsittelyä erilaiseen muotoon. Sisällön tallennus tietovarantoon voi kuitenkin muuttua tästä huomattavasti laajemmaksi ja monimutkaisemmaksi konseptiksi. Konsepti voi sisältää muun muassa seuraavanlaisia toimintoja: uuden sisällön luonti käyttäjän syöttämän tiedon perusteella, tiedon keräys sisällöksi jostakin ulkoisesta tietolähteestä, tiedon muotoilu sisällöksi sopivaan formaattiin, sisällön jako erilaisiin osakokonaisuuksiin ja sopiviin formaatteihin sekä erilaisten tietosisältöjen keruun aputoimintojen tuottaminen. [Boiko 2005] Aputoimintoja voivat olla esimerkiksi jonkinlaiset tiettyjen annettujen verkkosivujen tietoja automaattisesti yhteen keräävät toiminnot.

Toinen CMS-järjestelmien yleisistä konsepteista on kerätyn sisällön hallinta, johon näistä järjestelmistä käytetty nimityksin viittaa. Kuten sisällön keruussa myös sisällönhallinnassa toiminnallisuuden määrä vaihtelee suuresti sen mukaan, kuinka laajasta järjestelmästä on kyse. Sisällönhallinnan toiminnallisuuden konseptissa kaikista keskeisin osa on kuitenkin tuottaa tietoa saataville sisällön perustiedoista [Boiko 2005]. Tähän voi liittyä esimerkiksi tarve tietää mihin organisaation osaan jokin tietty sisältö liittyy [Barker 2016]. Muita tavallisesti sisällönhallintaan liittyviä toiminnallisuuden osia ovat: tietovarasto, johon sisältö tallennetaan sekä työkalut tallennetun sisällön hallintaan, CMS-järjestelmän yleiset hallintatoiminnot, uuden sisällön julkaisuprosessin askeleet sekä mahdolliset yhteydet muihin integroituihin järjestelmiin esimerkiksi organisaation sisällä sisällön hallitsemiseksi [Boiko 2005]. Laajassa järjestelmän versiossa CMS-järjestelmää voidaan käyttää yhtenä linkkinä, josta sisältö jaetaan hyvin moneen erilaiseen muuhun järjestelmään automaattisesti [Boiko 2001].

Kolmas CMS-järjestelmien yleinen konsepti koskee kerätyn sisällön julkaisun hallintaa. Julkaisutoimintojen keskeinen tarkoitus on ottaa tietovarastoon tallennettu sisältö ja tuottaa siitä halutunlainen julkaisu [Boiko 2005]. Sopiva julkaisumuoto riippuu

täysin siitä, minkälaiseen ympäristöön sisältö ollaan julkaisemassa. Esimerkiksi alaluvussa 2.1 käsiteltävien web CMS -järjestelmien tapauksessa sopiva julkaisumuoto voisi olla HTML-sivu. Julkaisutoiminnoissa saattaa olla mukana esimerkiksi seuraavia toiminnallisuuden osioita: julkaisuja automaattisesti generoivia työkaluja tai pohjia, valintamahdollisuudet sille mitä ja miten julkaistaan, linkit mahdollisiin julkaisuissa käytettyihin muihin resursseihin sekä linkit sopiviin julkaisualustoihin, joihin julkaistu sisältö lähetetään [Boiko 2005].

Vaikka CMS-järjestelmien kolme toiminnallisuuksien konseptia kuvaavatkin erilaisia järjestelmistä löytyviä toiminnallisuuksia, ei ole olemassa yhtä tiettyä rakenteellista ja toiminnallisuuksia kuvaavaa eksplisiittistä mallia, joka sopisi kaikkiin erilaisiin CMS-järjestelmiin. Nämä järjestelmät ovat syntyneet hyvin moninaisista ja erilaisista tarpeista kommunikoida eri tahojen välillä sisällönhallinnan keinoin [Boiko 2005]. CMS-järjestelmien merkityskin riippuu voimakkaasti siitä mistä näkökulmista järjestelmiä tarkkaillaan. Esimerkiksi yrityksen mainonnan ja markkinoinnin kannalta kaiken markkinointimateriaalin järjestelmällinen hallinta on keskeistä.

CMS-järjestelmiä voidaan jaotella myös sen mukaisesti, kuinka laajoja järjestelmiä ne ovat. Yksinkertaisimmillaan CMS-järjestelmä voi toimia lähinnä verkkosivuston tavallisena ylläpitotyökaluna [Boiko 2001]. Tällaisia järjestelmiä onkin saatavilla todella paljon esimerkiksi SAAS-pohjaisina palveluina. Laajimmat CMS-järjestelmät ovat kuitenkin huomattavasti suurempia ja esimerkiksi *yritysten sisällönhallintajärjestelmät* (Enterprise CMS) koostuvat hyvin monimutkaisista ja laajoista erilaisista toiminnoista. Jo 2000-luvun alkupuolelta lähtien tyypillisimmät CMS-järjestelmät ovat kuitenkin keskittyneet nimenomaan yritysten verkkosivujen ja niihin liittyvien erilaisten lisätoimintojen hallintaan [Boiko 2001]. Tämän tutkielman tulevissa luvuissa tullaankin keskittymään lähinnä web-pohjaisiin yritysten verkkosivujen hallintaan tarkoitettuihin CMS-järjestelmiin ja niiden erilaisiin toimintoihin sekä ohjelmistoteknisiin ratkaisuihin ja mahdollisiin rajoitteisiin.

2.1 Web CMS -järjestelmät

Tänä päivänä digitaalisen markkinoinnin perustan muodostaa yrityksen oma verkkosivusto [Grubor ja Jakša 2018]. Kaikilla yrityksillä ei ole kuitenkaan taloudellisia resursseja monimutkaisten sivustojen teettämiseen tai toisaalta tarvittavaa tietotaitoa web-ohjelmoinnista, jotta he voisivat itse rakentaa sivustonsa tyhjästä. Tässä kohdassa web-pohjaisten CMS-järjestelmien käyttö on hyvin yleistä ja tammikuussa 2018 jo yli 50 % verkkosivuista toteutettiin web CMS -järjestelmien avulla [Jose-Manuel *et al.* 2018]. Web CMS -järjestelmän kanssa käyttäjän ei yleensä tarvitse itse osata juurikaan web-ohjelmoinnin tekniikoita, vaan järjestelmä avustaa sisällön muotoilemisessa haluttuun muotoon [Jose-Manuel *et al.* 2018]. Saatavilla voi olla myös web CMS -järjestelmästä

riippuen paljon erilaisia helposti käyttöön valittavissa olevia tyylipohjien ja teemojen kokonaisuuksia, jolloin omien sivujen rakentaminen persoonalliseksi on myös helppoa.

Web CMS -järjestelmät eroavat tavallisista CMS-järjestelmistä erityisesti niiden käyttötarkoituksen osalta. Web CMS -järjestelmien pääfunktio on ennen kaikkea mahdollistaa sisällön hallinta ja tuottaminen verkkosivuilla julkaistavaksi suurelle kuluttajamäärälle [Barker 2016]. Web CMS -järjestelmät ovatkin niin yleisiä CMS-järjestelmien toteutusmuotoja, että joissakin yhteyksissä näistä puhutaankin samana asiana. Web CMS -järjestelmät ovat kuitenkin nähtävissä selkeästi yhdeksi CMS-järjestelmien tyypiksi ja niillä on nähtävissä oma vähitellen syvempään erikoistumiseen johtanut historiansa [Boiko 2001]. Esimerkiksi Vaidya *et al.* [2013] käsittelee virheellisesti CMS-järjestelmiä lähinnä web CMS -muotoisina järjestelminä; artikkeli jopa määrittelee CMS-järjestelmän keskeiseksi tehtäväksi näyttää informaatiota verkkosivuilla.

Web CMS -järjestelmät koostuvat yleensä kahdesta erilaisesta järjestelmän osiosta: käyttöoikeuksiltaan rajatusta hallintapuolesta (backend) sekä avoimesta julkisesta puolesta (frontend) [Jose-Manuel *et al.* 2018]. Hallintapuolen kautta sen käyttöön oikeutetut käyttäjät pääsevät muokkaamaan asetuksia ja luomaan sekä hallitsemaan järjestelmän avulla julkaistavaa sisältöä. Joissakin tapauksissa käyttöoikeudet voivat olla hyvinkin monipuolisia ja eri tasoisia [Jose-Manuel *et al.* 2018]. Tällöin web CMS -järjestelmä taipuu laajempienkin yritysten tai organisaatioiden käyttöön, jossa eri tasoiset käyttöoikeudet ovat keskeisiä.

Web CMS -järjestelmät sisältävät tyypillisesti myös tietynlaisia ominaisuuksia riippumatta järjestelmän laajuudesta [Boiko 2001]. Järjestelmissä pystytään yleensä luomaan jonkinlaisen WYSIWYG-editorin avulla HTML:stä koostuvia tietokantaan tallennettavia hierarkkisia sivuja [Vaidya *et al.* 2013, Boiko 2001]. Näiden sivujen keskinäisestä hierarkkisesta muodosta koostetaan sitten kutsujen yhteydessä käyttäjälle näytettävää sisältöä joidenkin ennalta määritettyjen tyylimuotojen avustuksella. Näiden ominaisuuksien tuottamiseksi web CMS -järjestelmissä on yleensä käytössä jonkinlainen tietovarastona toimiva tietokanta, joka voi olla esimerkiksi MySQL-pohjainen [Vaidya *et al.* 2013]. Web CMS -järjestelmien keskeinen tehtävä on myös toteuttaa tänä päivänä verkkosivuille keskeisiä muita tukitoimintoja, kuten antaa hyvät mahdollisuudet hakukoneoptimointiin [Jose-Manuel *et al.* 2018].

Web CMS -järjestelmät voidaan myös jakaa erilaisiin tasoihin riippuen niiden toiminnallisuuden laajuudesta. Yksinkertaisimmillaan *nimellinen web CMS -järjestelmä* (The nominal web CMS) sisältää mahdollisuudet HTML-sisällön rakentamiseen WYSIWYG-editorin avulla jonkinlaisen yksinkertaisen hallintapaneelin välityksellä. Tällöin lopullisena kohteena on yleensä hyvin pienen tiimin tai yksittäisen henkilön

ylläpitämä yksittäinen lähes staattinen verkkosivu, johon kaikki luotu sisältö kohdennetaan. [Boiko 2001]

Toinen web CMS -järjestelmien taso on *dynaaminen verkkosivu* (dynamic website), jossa olemassa oleviin tietynlaisiin rakenteisiin tuotetaan dynaamisesti sisältöä käyttäjän tekemien kutsujen perusteella. Tällöin sisältö on muodostettu usein ennalta hallintapaneelin kautta ja järjestelmä saattaa mahdollistaa esimerkiksi sisällön personoinnin käyttäjäkohtaisesti. Hyvä web CMS -järjestelmä antaa kuitenkin yleensä dynaamisia verkkosivuja monipuolisemmat mahdollisuudet tiedon keräämiseen ja muotoiluun sopivaksi sisällöksi, paremmat mahdollisuudet sisällön hallintaan sekä enemmän tietoa sisällön muodosta kullakin hetkellä. Lisäksi on hyvä huomata, että verkkosivu voi olla dynaaminen ilman, että se tarjoaa mitään sisällön hallintaan liittyviä toimintoja. Tällöin kyseessä ei ole varsinaisesti edes CMS-järjestelmä, vaan sinällään tavallinen verkkosivusto. [Boiko 2001]

Ylivoimaisesti suosituin web CMS -järjestelmä on avoimeen lähdekoodiin pohjautuva WordPress. Kaksi muuta niin ikään hyvin suosittua, mutta WordPressiin nähden marginaalisempaa web CMS -järjestelmää ovat myös avoimeen lähdekoodiin pohjautuvat Joomla! ja Drupal. [Jose-Manuel *et al.* 2018] Nämä kaikki kolme web CMS -järjestelmää kuuluvat käytännössä vaihtelevasta ominaisuuksien kirjostaan riippumatta web CMS -järjestelmien kolmannelle tasolle eli ne ovat *täysimääräisiä web CMS -järjestelmiä* (The full web CMS). Näille web CMS -järjestelmille tyypillistä on, että järjestelmä on jaettu hyvin selkeästi julkiseen sekä käyttöoikeuksien osalta rajattuun hallintapuoleen, jossa järjestelmän keskeisiä ylläpitotoimia voidaan suorittaa. Hallintapuolella näissä järjestelmissä on tarvittavat työkalut, joilla sivustoa voidaan rakentaa ja ylläpitää, kuten erilaiset sisältö- ja tyylieditorit sekä näkyvyyksien ja julkaisutoimien hallinnat. Näissä järjestelmissä on usein pohjalla jonkinlainen joukko kiinteitä HTML-sivuja, joiden päälle varsinainen sisältö sitten rakennetaan kutsujen yhteydessä. [Boiko 2001]

Nämä tässä esitellyt täysimääräisten web CMS -järjestelmien toiminnot ovat kuitenkin pääpiirteittäin sellaisia, joita esimerkiksi dynaamiset verkkosivut ja osittain myös nimelliset web CMS -järjestelmät toteuttavat. Erittäin keskeinen ero näihin kahteen tyyppiin onkin täysimääräisille web CMS -järjestelmille tyypillinen integraatio muihin erilaisiin järjestelmiin [Boiko 2001]. Laajimmillaan web CMS -järjestelmä ei toimi tällöin välttämättä edes pelkästään yhden sivuston rakennukseen käytettynä hallintatyökaluna, vaan sitä voidaan käyttää useiden erilaisten sivustojen hallintaan tarkoitettuna erilaisten sisältömoduulien tai muiden tietolähteiden ylläpitotyökaluna, josta sisältöä sitten haetaan sopiviin sovelluskohteisiin [Boiko 2001].

Täysimääräiseen web CMS -järjestelmään voidaan liittää erilaisia muita palveluita ja toimintoja. Yleinen lisäosa näissä järjestelmissä onkin esimerkiksi

verkkokaupan integrointi järjestelmään [Boiko 2001]. Esimerkiksi WordPress-pohjaisiin järjestelmiin on mahdollista lisätä hyvin monipuolisesti erilaisia verkkokauppa-alustoja järjestelmän lisäosina, jotka mahdollistavat muun muassa tavallisen verkkokaupan tai jopa verkkokaupan ja varausjärjestelmän yhtäaikaisen ja saumattoman ylläpidon itse verkkosivuston yhteydessä [Pinpoint 2019]. Kun erilaisten järjestelmään lisättyjen lisäominaisuuksien, integraatioiden, toimintojen tasojen ja ylipäätään kompleksisuuden määrä lisääntyy, kasvaa luonnollisesti myös järjestelmän vaatima laskentatehon määrä, mikä on nähtävissä esimerkiksi Drupalin ja Joomla!:n suorituskyvyssä [Ogunrinde ja Yoosuf 2016].

Kun web CMS -järjestelmään lisätään paljon käyttäjäkohtaisesti personoitua sisältöä, tarkoittaa se yleensä sitä, että jokaisen sivun latauksen yhteydessä tehtävän käsittelyn määrä kasvaa huomattavasti [Boiko 2001]. Oikeastaan hyvin harvoin web CMS -järjestelmien tuottamat sivukokonaisuudet tehdään täysin staattisten resurssien perusteella. Jos personoitua sisältöä halutaan käyttää, tarkoittaa se aina sitä, että jonkinlaista prosessointia on pakko tehdä sen sijaan, että näytettäisiin esimerkiksi vain staattisia HTML-sivuiksi tallennettuja kokonaisuuksia [Boiko 2001]. Tämä korostuukin näiden järjestelmien suoritusarvoissa ja esimerkiksi Drupalin ja Joomla!:n tapauksessa suoritusarvoja pyritäänkin parantamaan yleisesti hyödyntämällä erilaisia välimuisteihin perustuvia ratkaisuja [Ogunrinde ja Yoosuf 2016]. Näitä erilaisia ohjelmistoteknisiä ratkaisuja esitellään tarkemmin seuraavassa alaluvussa 2.2. Modernit web CMS -järjestelmät ovat siis yleensä niin sanottuja dynaamista sisältöä rakentavia järjestelmiä. Vastaavasti jokin erikseen esimerkiksi staattiseksi HTML-sivukokonaisuudeksi web CMS -järjestelmän avulla luotu kokonaisuus, jossa dataa ei prosessoida erikseen kutsujen yhteydessä, olisi staattista sisältöä näyttävä järjestelmä [Boiko 2001].

2.2 Dynaamisten web CMS -järjestelmien ohjelmistotekniset ratkaisut

Web CMS -järjestelmät ovat sinänsä tavallisia web-pohjaisia tietojärjestelmiä, joten niissä käytetään myös yleisesti web-ohjelmoinnissa käytettyjä tekniikoita, kuten HTML5:sta ja CSS3:sta. Nykyään dynaamisten, eli täysimääräisten, web CMS -järjestelmien tapauksissa tarvitaan kuitenkin myös jollain asteella dynaamista kutsujen yhteydessä tapahtuvaa sisällön käsittelyä ja prosessointia, mikä onnistuu useimmissa tapauksissa PHP:n avulla [Jose-Manuel *et al.* 2018]. PHP:n lisäksi dynaamisen web-ohjelmoinnin pohjana voidaan kuitenkin käyttää myös muita ratkaisuja, kuten esimerkiksi .net ja Java EE -ratkaisuja [Komara *et al.* 2016].

Suosituimmat web CMS -järjestelmät tukevat muutamaa erilaista ratkaisua järjestelmän käyttämän tietovarannon osalta [Jose-Manuel *et al.* 2018]. Web-ohjelmoinnille tyypillisten tekniikoiden tavoin useimmiten tuettuna on jonkinlainen joukko erilaisia SQL-tietokantoja, joista valittuun tietokantaan järjestelmän tuottama sisältö ja sisällön hallinnollinen tieto tallennetaan. Esimerkiksi MySQL on hyvin usein

käytetty ratkaisu [Jose-Manuel *et al.* 2018]. Aiemmin alaluvussa 2.1. pintapuolisesti esitellyissä suosituissa web CMS -järjestelmissä, eli WordPressissä, Joomla!:ssa ja Drupalissa, on kuitenkin tuki muutamille erilaisille tietokantaratkaisuille, joista järjestelmän asennuksesta vastaava taho voi valita parhaiten tilanteeseen sopivan.

Viime aikoina myös esimerkiksi Node.js-pohjaisten järjestelmien arkkitehtuurien rakennusta on testattu hyvin tuloksin. Kaimer ja Brune [2018] testasivat ohjelmistoarkkitehtuuria, joka oli suunniteltu juuri web CMS- ja customer relationship management -järjestelmiä varten palvelinpuolen Node.js:n Sails.js-kirjaston ja selainpuolen JavaScript-kirjaston Vue.js:n avulla. Heidän mukaansa tämä yhdistelmä tarjoaa varsin suorituskykyisen ja turvallisen toimintaympäristön web CMS -järjestelmän tarpeisiin. Näiden vaihtoehtojen kehittyessä ja tekniikoiden vakiintuessa voisi olettaa uusiin tekniikoihin pohjautuvien järjestelmien myös ilmestyvän vähitellen. Tätä tukee myös Node.js:n ehdoton valttikortti suhteessa PHP-pohjaisiin ratkaisuihin: Node.js pystyy hyödyntämään kaikki saatavilla olevat prosessorin suoritusytimet, kun taas esimerkiksi PHP pystyy hyödyntämään vain yhden [Chaniotis *et al.* 2015].

Web CMS -järjestelmiltä odotetaan muiden rikkaiden Internet-sovellusten tavoin varsin hyvällä tasolla olevaa suorituskykyä [Ying ja Miller 2012]. Web CMS -järjestelmät ovat kuitenkin usein varsin laajoja tietojärjestelmiä, jolloin jokaisen kutsun yhteydessä suoritetaan aina suurehko määrä käsittelyä ja prosessointia, minkä seurauksena kutsujen suoritusajat kasvavat väistämättä suuremmiksi [Boiko 2001]. Osa tästä käsittelystä ja prosessoinnista voi olla käsiteltävälle kutsulle turhaa, ja esimerkiksi MVC-mallia seuraavissa järjestelmän kehyksissä on väistämättä suoritusta hidastava suuri järjestelmän ydin [Wang 2011]. Yleensä jokainen tällaiseen laajaan web CMS -järjestelmään kohdistuva kutsu käynnistää järjestelmän ytimen, minkä suoritus vie yleensä aina vähintään tietyn verran suoritustaikaa [Wang 2011]. MVC-mallin seuraamisella on kuitenkin myös hyvät puolensa: se tarjoaa erittäin tarkan jaon järjestelmän erilaisten osien välille [Wang 2011]. Tällöin esimerkiksi järjestelmän ylläpito voi olla suoraviivaisempaa.

Moni web CMS -järjestelmissä tehtävä toiminto nojaa ainakin osittain asynkronisiin kutsuihin esimerkiksi AJAX:n avulla toteutettuna. Jotta järjestelmien käyttö olisi helppoa ja käyttäjäystävällistä, tulee AJAX-kutsujen olla erittäin nopeita [Ying ja Miller 2012]. Jotta kutsut voisivat olla nopeita, pitäisi suurten järjestelmien suorituksen olla nopeaa yksittäisten hyvin yksinkertaistenkin kutsujen yhteydessä ilman järjestelmän käyttämien resurssien saannin vaarantumista esimerkiksi tietoturvan näkökulmasta. Tätä varten on testattu esimerkiksi toisen kevennetyn suorituspolun rakentamista järjestelmän sisälle alkuperäisen raskaan ja paljon erilaisia resursseja käyttävän ytimen tilalle [Cu *et al.* 2009]. Tällöin ongelmaksi muodostuu kuitenkin edelleen kasvava koodipohjan ylläpidon määrä, mikä ei välttämättä tue järjestelmän rakennusta toivottuun suuntaan kehittäjien kasvavan taakan myötä [Nederlof *et al.* 2014].

Jotta web CMS -järjestelmissä voitaisiin jollain tavalla kiertää mahdollisia järjestelmien suurista ja kompleksisista perusrakenteista johtuvia suorituskyvyn ongelmia, on niissä usein käytössä välimuistin käyttöön pohjautuvia ratkaisuja [Ogunrinde ja Yoosuf 2016]. Välimuisti voi koostua useista erilaisista tasoista ja sinne voidaan ladata oikeastaan minkälaista tietoa hyvänsä, mikä edesauttaa toistuvien kutsujen käsittelyä, kun tarvittava tieto on jo jossain määrin olemassa ja valmiiksi käsiteltynä. Esimerkiksi PHP:n tapauksessa keskeisenä välimuistina toimii käynnissä oleva sessio, johon tietoa voidaan tallentaa myöhemmin käytettäväksi [PHP Sessions 2019]. Session käytön huonona puolena on järjestelmän käyttämän muistin määrän kasvu, mutta se on yleensä kuitenkin vähemmän ongelmia aiheuttava tekijä kuin kaikkien käytettävien tietojen haku uudestaan ja uudestaan kutsujen yhteydessä [Sa'adah *et al.* 2015]

Välimuistin käyttö tuottaakin erittäin merkittäviä tuloksia myös suorituskyvyn parantamisen näkökulmasta. Esimerkiksi Joomla!:n ja Drupalin tapauksessa sivujen latausaikoja on testattu ja ne ovat merkittävästi pienempiä silloin, kun osa käytettävistä tiedoista on tallennettuna välimuistiin [Ogurinde ja Yosuuf 2016]. PHP:n session välimuistin lisäksi voidaan käyttää erilaisia välimuisteja esimerkiksi tietokantahakujen tuloksien varastointiin I/O-operaatioiden vähentämiseksi [Sa'adah *et al.* 2015]. Lisäksi voidaan käyttää myös uudehkoa "service worker" -tekniikkaa, jolloin PHP-koodin suoritusta ei tarvitse edes välttämättä tehdä ollenkaan toistuvien kutsujen tapauksessa selaimen palauttaessa välimuistiin tallennetun sivun suoraan [Amarasinghe 2016]. PHP:n yhteydessä voidaan käyttää myös PHP:n laajennosta OPcachea välimuistina, jolloin PHP-koodin käännetty versio suoritetaan uudestaan kutsun yhteydessä ilman uutta kääntämistä, jos alkuperäinen kääntämätön tiedosto ei ole muuttunut [Arsenault 2017].

Varsin moni erilaisista ohjelmistoteknisistä ratkaisuista suosituissa web CMS -järjestelmissä tähtää juurikin näille järjestelmille muiden web-pohjaisten järjestelmien tavoin keskeisten tavoitteiden saavuttamiseen: suorituskyvyn, luotettavuuden ja saavutettavuuden parantamiseen. Tällöin latausajat saadaan alas sekä saavutetaan parempi käyttökokemus ja tyytyväisemmät asiakkaat [Laird ja Brennan 2006]. Suosituissa PHP-pohjaisissa web CMS -järjestelmissä myös PHP itsessään aiheuttaa erilaisia rajoitteita järjestelmän toiminnalle ja sen suorituskyvylle edellä esiteltyjen erilaisten ongelmien ja ratkaisumallien lisäksi.

2.3 PHP:n piirteistä ja rajoitteista web CMS -järjestelmissä

Suosituimmat web CMS -järjestelmät WordPress, Joomla! ja Drupal ovat PHP-pohjaisia [Jose Manuel *et al.* 2018]. Lisäksi muita PHP:lle pohjautuvia hyvin suuria järjestelmiä ovat esimerkiksi Wikipedia ja Facebook [Hauzar ja Kofroň 2014]. PHP:tä ei ole kuitenkaan koskaan suunniteltu käytettäväksi näin laajoissa tietojärjestelmissä [Chaniotis *et al.* 2015]. Se on tarkoitettu lähinnä yksinkertaisilla vähän laskentaa vaativilla sivustoilla käytettäväksi, mihin kielen nimityskin viittaa: "Personal home page"

[Ruohonen *et al.* 2016]. Tässä alaluvussa käsitelläänkin erilaisia seikkoja, joiden kautta PHP aiheuttaa rajoitteita web CMS -järjestelmien ja muiden laajojen web-pohjaisten tietojärjestelmien kehitykselle ja suorituskyvyn huomioinnille.

PHP on korkean tason kieli, jonka lähestyminen on erittäin helppoa. PHP:n suorittamiseksi ei tarvitse kovin monimutkaista web-ympäristöä eikä ohjelmakoodia tarvitse kääntää itse ennen sen suoritusta. PHP:n kääntäminen tapahtuu yleensä vasta suorituksen yhteydessä eikä sitä käyttävän henkilön tarvitse tehdä kääntämistä varten mitään erillisiä toimenpiteitä. Tämä aiheuttaa kuitenkin sen, että kohtuullisen hyvästä suoritusnopeudestaan huolimatta PHP ei ole yhtä hyvin optimoitu kieli kuin esikäännetyt ohjelmointikielet ovat [Cholakov 2008].

Jotta PHP toimisi halutulla tavalla suorituksen yhteydessä käännettävänä kielenä, on PHP-pohjaisille tietojärjestelmille tyypillistä ohjelmistoarkkitehtuuri, jossa juuri mitään laskentaa tai käsittelyä ei ole jaettu eri suorituskertojen kesken [Popov *et al.* 2017]. Tämän takia suurin osa kutsuista PHP-pohjaisiin järjestelmiin aloitetaan aina ikään kuin tyhjästä, jolloin kaikki tarvittavat resurssit haetaan aina uudestaan suorituskertojen välillä [Popov *et al.* 2017]. Osittain tästä johtuen esimerkiksi Joomla!:n ja Drupalin latausajat ovat huomattavasti pienempiä silloin, kun jonkinlaista välimuistia on käytetty osana järjestelmään kohdennettujen kutsujen suoritusta [Ogunrinde ja Yoosuf 2016].

Jokaisen suorituksen yhteydessä tehtävän käännöksen lisäksi PHP:n suorituskykyä ja skaalautuvuutta rajoittaa suhteessa uudempiin web-tekniikoihin sen tuki vain yhdelle suoritinytimelle [Chaniotis *et al.* 2015]. Samalla PHP-pohjaisten järjestelmien I/O-operaatioiden käsittelynopeus ei ole myöskään yhtä hyvällä tasolla suhteessa esimerkiksi Node.js:n saavuttamiin tuloksiin [Chaniotis *et al.* 2015]. Tämä aiheuttaa sen, että esimerkiksi ulkoiseen tietokantaan tehtävät runsaat kutsut voivat laskea PHP-pohjaisten järjestelmien suorituskykyä merkittävästi.

PHP:n maine ohjelmointikielenä ei ole kovin imarteleva. Sitä pidetään yleisesti helposti lähestyttävänä kielenä, mutta käytännöiltään varsin huonona. Osasyynä tälle on PHP:n mahdollistama hyvin vapaa muuttujien tyypitys sekä esimerkiksi erilaisten globaalien muuttujien käytön rajoittamattomuus [Cholakov 2008]. Lisäksi osa PHP:n omista funktioista on nimetty jossain määrin vaihtelevin käytännöin tehden sopivan funktion valinnan paikoitellen hankalaksi [Cholakov 2008]. Osittain näistä syistä johtuen PHP:n mahdollistamaa ohjelmakoodin perusrakennetta on kuvattu esimerkiksi hyvin virhealttiiksi ja epätehokkaaksi [Hauzar ja Kofroň 2014].

PHP:n huono maine voi johtua osittain siitä, että sitä on erittäin helppo käyttää väärin. Kieli antaa paljon virheitä anteeksi, ja esimerkiksi muuttujien käytön osalta vähemmän kokenut kehittäjä voi tehdä sellaisia virheitä, joiden löytäminen jälkeenpäin voi olla erittäin hankalaa [Cholakov 2008]. Kun vastuu hyvän ohjelmakoodin kirjoittamisesta on kielen toimintalogiikan vuoksi kehittäjän vastuulla, on uusien

kokeneidenkin kehittäjien hankalampi rakentaa laadukasta ohjelmakoodia jo olemassa oleviin järjestelmiin, jos he eivät jo entuudestaan tunne järjestelmää.

PHP:n kehityksessä nämä erilaiset tarpeet on kuitenkin tunnistettu ja aiemmin ongelmalliseksi koettuihin seikkoihin on pyritty tuomaan ratkaisukeinoja PHP:n uudemmissa versioissa [Popov *et al.* 2017]. Esimerkiksi tyypitykseen on tuotu uusia mahdollisuuksia PHP:n versiossa 7 sekä sen jälkeisissä versioissa [Popov *et al.* 2017]. Toisaalta samalla PHP:n uusille versioille tyypilliset suuret muutokset voivat myös aiheuttaa ongelmia vanhempien järjestelmien yhteensopivuuden kanssa [Cholakov 2008]. Tämä voi aiheuttaa merkittäviä ongelmia esimerkiksi vanhemmille web CMS -järjestelmille, kun niiden rakenteita joudutaan muuttamaan uusien PHP-versioiden käyttöönoton yhteydessä. Samalla uudemmat PHP:n julkaisut ovat kuitenkin tuoneet myös merkittäviä hyppyjä suorituskyvyn osalta [Gavalda 2019]. Web CMS -järjestelmille onkin tärkeää pysyä uusimmissa ohjelmakoodin versioissa myös suorituskyvyn näkökulmasta.

Web CMS -järjestelmien kehityksessä on hankalaa käyttää PHP:n yhteydessä ohjelmistokehityksessä muutoin käytettyjä ohjelmakoodin rakennetta ja mahdollisia ongelmakohtia kartoittavia työkaluja, kuten staattista analyysiä, sen dynaamisen luonteen vuoksi [Hauzar ja Kofroň 2014]. PHP:n suorituksen analysoinnissa voikin käyttää tehokkaammin hieman erilaisia ratkaisuja, kuten PHP:n laajennosta Xdebugia, jonka avulla voi esimerkiksi profiloida ohjelmakoodin suorituksia ja siten vertailla esimerkiksi eri funktioiden kuluttamaa suoritusaikaa sekä löytää piileviä ongelmakohtia [Xdebug 2019]. Staattisen analyysin ongelmallisuuden vuoksi sen keinoja käytetäänkin PHP:n yhteydessä enemmän suoritusympäristön kuin itse ohjelmakoodin suorituskyvyn parantamiseen [Popov *et al.* 2017].

Käytettäessä PHP:tä web CMS -järjestelmän pohjana on pitkäjänteisen tuotekehityksen kannalta kuitenkin kaikista keskeisintä pitää huoli siitä, ettei kielen anteeksiantavaisuuden ja monipuolisuuden anneta tehdä rakennettavasta järjestelmästä hankalasti ylläpidettävää, epäselvää ja virheherkkää kokonaisuutta. Mikäli näistä seikoista ei pidetä huolta, voivat kehityskulut ja niiden vaatima työaika kasvaa sekä vikojen esiintyvyyden määrä kasvaa [Rocha *et al.* 2017]. Tämän huomiointi on erityisen tärkeää varsinkin vanhempien järjestelmien yhteydessä, koska PHP:n rakenne itsessään on muotoutunut vuosien varrella useiden eri versioiden myötä [Cholakov 2008].

Jotta PHP:llä toteutettu web CMS -järjestelmä säilyisi edelleen eheänä kokonaisuutena useiden järjestelmän iteraatioiden myötä, on tärkeää, että järjestelmän laatutaso pystytään säilyttämään ja seuraamaan sitä aktiivisesti osana järjestelmän normaalia kehitystä. Luvussa 3 käsitellään ohjelmistojen laatuun liittyviä erinäisiä tekijöitä, miten laatua voidaan mitata ja arvioida sekä millaisia ilmiöitä laatuun osana jatkuvaa ohjelmistojen kehitystä liittyy.

3 Ohjelmiston laatu, laadun arviointi ja metriikat

Ohjelmiston laatu on varsin laaja käsite, ja jonkin ohjelmistoa käyttävän henkilön subjektiivisen laatuvaikutelman muodostumiseen vaikuttavat monet erilaiset seikat. Käyttäjän aiemmat kokemukset, tietotaito aihepiiristä sekä erilaiset muut tekijät vaikuttavat suuresti siihen, miten laatu koetaan. Jokin ohjelmisto voidaan mieltää laadukkaaksi esimerkiksi sen kautta, toimiiko se oletetulla tavalla ja pystyykö sitä käyttämään helposti mobiililaitteella. Toisaalta jossakin muussa yhteydessä laadukkaana ohjelmistona voitaisiin pitää sellaista ohjelmistoa, joka pystyy suoriutumaan siihen tehdyistä kutsuista hyvin nopeasti eli suorituskykyisesti. Laatuvaikutelma on siis aina subjektiivinen kokemus, jonka arviointi ja mittaaminen on erittäin hankalaa ja epävarmaa. [Sommerville 2007]

Jotta laatua voisi arvioida sekä mitata ja jotta sen voisi ylipäätään määritellä objektiivisesti, tulee ohjelmistolle asettaa objektiivisesta näkökulmasta vaatimukset sekä laatutavoitteet. Olennaista on, että ollakseen laadukas tulisi ohjelmiston täyttää sille asetetut vaatimukset. Vaatimusten lisäksi laadukas ohjelmisto täyttää sille asetettuja erilaisia laatuominaisuuksia jollakin sopivalla tasolla. Kaikkea ei yleensä voida saavuttaa ja mikään ohjelmisto ei voi olla täydellinen. Tällöin onkin tärkeä asettaa laatuominaisuuksille sellainen vaatimustaso, joka on oikeasti saavutettavissa ja ei toisaalta syö muiden laatuominaisuuksien toteutusmahdollisuuksia tai nosta ohjelmiston kehitysbudjettia liian suureksi. Esimerkiksi ohjelmiston resurssien tehokas käyttö on laatuominaisuutena sellainen, että se tulee huomioida oikeastaan lähes kaikissa ohjelmiston osissa, jolloin se asettaa luonnollisesti huomioon otettavia rajoitteita ja tarpeita kaikkien muiden ominaisuuksien ja rakenteiden toteutukseen. [Wiegers ja Beatty 2013]

Ohjelmiston laatuominaisuudet voivat olla ulkoisia laatuun liittyviä asioita, kuten saavutettavuus, suorituskyky, turvallisuus ja käytettävyys tai sisäisiä, kuten uudelleenkäytettävyys ja skaalautuvuus. Ulkoiset laatuominaisuudet ovat sellaisia seikkoja, joita tarkkaillaan ohjelmiston suorituksen yhteydessä ja joiden tason kuka tahansa ohjelmistoa käyttävä taho voi kokea jollakin tasolla. Sisäiset laatuominaisuudet ovat taas vastaavasti sellaisia ominaisuuksia, jotka eivät ole suoraan havaittavissa ohjelmiston suorituksen yhteydessä. Nämä ominaisuudet ovat pikemminkin sellaisia, joita järjestelmää kehittävä ja ylläpitävä taho pitää silmällä esimerkiksi ohjelmiston arkkitehtuurin ja yksittäisten ratkaisujen rakenteiden tarkkailun kautta. Sisäiset laatuominaisuudet voivat kuitenkin näkyä myös ulospäin esimerkiksi sellaisissa tilanteissa, joissa ne rajoittavat keinoja, joilla ohjelmistoa voidaan jatkokehittää tai nostavat kustannuksia huomattavasti suuremmiksi. [Wiegers ja Beatty 2013]

Tässä tutkielmassa keskitytään lähtökohtaisesti laatuun suorituskyvyn näkökulmasta ja tämän luvun tulevissa osissa nostetaankin esimerkkejä erityisesti

suorituskykyyn liittyvien laatuominaisuuksien kautta. Suorituskyvyn osalta laatu voidaan jakaa sekä ulkoiseen että sisäiseen puoleen. Ulkoisena tekijänä suorituskyvyllä tarkoitetaan järjestelmän kykyä reagoida siihen kohdistuneisiin käyttäjän kutsuihin ja pyyntöihin. Tällöin laatua voidaan tarkastella esimerkiksi kutsujen suoritusajan, määrältään suurten yhtäaikaisten kutsujen käsittelykyvyn ja kutsujen viiveaikojen kautta. Sisäisenä tekijänä suorituskyvyllä viitataan järjestelmän kykyyn olla olemassa olevien resurssien käytön osalta tehokas. [Wiegers ja Beatty 2013]

Sisäiseltä suorituskyvyltään laadukas, eli tehokas, ohjelmisto pyrkii tekemään mahdollisimman vähän turhia suorituksia ja toisaalta tekemään tarvittavat suoritukset mahdollisimman tehokkaasti [Wiegers ja Beatty 2013]. Tämä muodostuu kuitenkin helposti ongelmaksi, sillä suorituskykyyn liittyviä laatuominaisuuksia ei yleensä nosteta kovin korkealle prioriteetille järjestelmiä kehitettäessä ja niiden vaatimuksia määriteltäessä [Alcocer ja Bergel 2015]. Tällöin mahdolliset järjestelmään syntyneet laatua laskevat ratkaisut voivat muotoutua ajan saatossa erittäin hankaliksi löytää ja poistaa [Alcocer ja Bergel 2015].

Dynaamisten web-ohjelmistojen tapauksissa järjestelmiä rakentavien ja ylläpitävien kehittäjien tulee yleensä tuottaa hyvin korkealaatuisia loppuratkaisuja varsin lyhyessä ajassa [Komara *et al.* 2016]. Ratkaisuille asetetut vaatimukset ja laatutavoitteet ovat yleensä teknisen toteutuksen näkökulmasta hyvin kalliita toteuttaa ja vaativat kehittäjiltä hyvin monipuolista osaamista [Komara *et al.* 2016]. Esimerkiksi suorituskyvyn osalta odotetaan, että järjestelmät ovat käytännössä aina saatavilla ja vastaavat käytännössä heti kutsuihin [Ying ja Miller 2012]. Nämä tavoitteet voivat olla hyvin vaikeita saavuttaa, jos järjestelmältä odotetulle laadulle ei ole varattu sen mahdollistamiseksi tarvittua kehitysaikaa. Tällöin lopputulemana saattaa olla se, että web-ohjelmiston kehityksessä tehdään laadun näkökulmasta haitallisia oikomisia, jotta lyhyen tähtäimen taloudelliset ja aikataululliset tavoitteet pystytään saavuttamaan [Rocha *et al.* 2017]. Ehkä osittain tämänkin takia web-pohjaisilla järjestelmillä on teknisestä näkökulmasta yleisesti varsin kyseenalainen maine niissä käytettyjen ratkaisujen laadukkuuden osalta [Nederlof *et al.* 2014].

Vaikka uuden ohjelmiston kehityksessä olisikin tehty paljon työtä toivotun laatutason saavuttamiseksi ohjelmiston ensimmäisen version kohdalla, ei laatu välttämättä säily ohjelmiston koko elinkaaren ajan [Eick *et al.* 2001]. Ilman jatkuvia toimenpiteitä halutun laatutason säilyttämiseksi ohjelmiston laatutaso alkaa vähitellen heikentyä koodipohjan muutosten myötä [Eick *et al.* 2001]. Laadun huomiointi osana ohjelmiston elinkaarta onkin äärimmäisen tärkeää, ja sen tulisi olla luonnollinen osa ohjelmiston tuotantolinjaa.

3.1 Laatu osana ohjelmiston elinkaarta

Ohjelmiston elinkaareen kuuluu luonnollisena osana ohjelmiston kehittyminen ja sen muuttuminen vastaamaan paremmin sille asetettuja ja uusia esiin nousevia vaatimuksia. Muutoin ohjelmisto muuttuu vähitellen yhä huonommin ympäristönsä tarpeita täyttäväksi [Alcocer ja Bergel 2015]. Ajan kuluessa ohjelmiston sisälle rakentuukin vähitellen monen muotoisia kerrostumia, jotka koostuvat sekoituksista eri ikäistä ohjelmakoodia. Ohjelmakoodi on yleensä rakennettu vastaamaan kirjoitushetkellä parhaita kehittäjän tuntemia menetelmiä ja ohjelmakoodin kielelle tyypillisiä parhaita käytäntöjä [Griffith *et al.* 2011]. Jos ei kuitenkaan kiinnitetä samalla huomiota ohjelmiston laatuun osana sen elinkaarta, tulee laatutaso väistämättä vähitellen laskemaan [Sommerville 2007]. Tällöin voi lopputulemana olla epämääräinen kokonaisuus, jonka tulkinta ja jatkokehittäminen on äärimmäisen hankalaa [Hassaine *et al.* 2012].

Vähitellen ohjelmakoodin muutosten myötä ohjelmiston alkuperäinen arkkitehtuuri, rakenne ja koodin yhtenäisyys heikkenevät, jolloin edellytykset koko ohjelmiston menestyksekkäälle kehittämiselle rapautuvat [Fowler *et al.* 1999]. *Ohjelmiston rappeutuminen* (software decay) onkin merkittävä haaste ohjelmiston kehitykselle [Eick *et al* 2001]. Ohjelmakoodia voidaan pitää rappeutuneena, kun sen muuttaminen on hankalampaa kuin sen pitäisi olla [Eick *et al.* 2001]. Tällöin pientenkin muutosten tekeminen onnistuneesti voi olla hyvin haastavaa. On kuitenkin tärkeää huomata, että ohjelmistossa vain osa sen ohjelmakoodista voi olla rappeutunutta samalla, kun osa ohjelmakoodista on kaikkea muuta kuin rappeutunutta. Onkin olennaista tiedostaa riskitekijät, jotka altistavat ohjelmiston rappeutumiselle.

Ohjelmiston rappeutumiselle tyypillisiä riskitekijöitä ovat ohjelmiston ikä, sisäinen kompleksisuus, kehitysorganisaatiosta johtuva paine, uudelleenkäytetty ohjelmakoodi, suuri vaatimusten määrä ja kehittäjät, joilla on vähän kokemusta tai jotka tuntevat järjestelmän huonosti [Eick *et al.* 2001]. Esimerkiksi tunnetuissa web CMS -järjestelmissä WordPressissä, Joomla!:ssa ja Drupalissa realisoituvia riskitekijöitä ovat ainakin näiden järjestelmien ikä, suuri vaatimusten määrä sekä huonosti järjestelmää tuntevat tai vähän kokemusta omaavat kehittäjät, sillä nämä ovat avoimeen lähdekoodiin pohjautuvia järjestelmiä.

Ohjelmiston rappeutuminen aiheuttaa yleensä monia erilaisia oireita, jotka on hyvä tunnistaa, koska ohjelmiston parissa ilmenevät erilaiset asiat voivat kieliä rappeutumisesta. Rappeutuneelle ohjelmistolle on tyypillistä, että sitä muokataan ja erityisesti korjataan usein. Mikäli tehtävät muutokset ovat vielä hyvin eritasoisia rakenteeltaan ja soveltuvuudeltaan ohjelmiston arkkitehtuuriin, voi kyseessä olla rappeutunut ohjelmisto. Rappeutuneesta ohjelmistosta kertoo myös muutos ohjelmiston ylläpidossa, joka muuttuu yhä hankalammaksi, kun ohjelmiston rakenne alkaa muuttua yhä monimutkaisemmaksi. Lisäksi oireita rappeutumisesta ovat muun muassa huonojen

korjausten tekeminen ja epämääräisyydet ohjelmiston toiminnassa sekä joissain tapauksissa myös useat erilaiset rajapinnat ohjelmiston eri osiin tai muihin järjestelmiin. [Eick *et al.* 2001]

Ohjelmiston rappeutumiseen ja siten sen laadun heikkenemiseen on useita erilaisia syitä. Teknisiä syitä ohjelmiston rappeutumiselle voivat olla esimerkiksi alkuperäisen arkkitehtuurin noudattamatta jättäminen muutoksia tehtäessä, alkuperäisen arkkitehtuurin huono tuki muutoksille, huonot työskentelyvälineet sekä epämääräiset vaatimukset, joiden seurauksena tuotetusta ohjelmakoodista ei tule tarpeeksi tasokasta. Kehitysorganisaatiosta johtuvia syitä ovat taas kehitysprosessiin liittyvät seikat, kuten huono muutostenhallintaprosessi sekä kommunikaation puute, huono yhteishenki, taloudelliset paineet ja osaamistason sekä ohjelmiston tuntemisen tason erot eri kehittäjien kesken. Kehittäjille ladattava aikataulupaine on myös erittäin keskeinen tekijä ohjelmiston rappeutumisessa. [Eick *et al.* 2001]

Kun kehittäjille annetaan hyvin tiukkoja aikatauluja, joihin pääseminen samalla annetut vaatimukset täyttäen voi olla erittäin hankalaa, jää usein vaihtoehdoksi jollain asteella heikompitasoisen ohjelmakoodin kirjoittaminen [Eick *et al.* 2001]. Tällöin kehittäjä saattaa tehdä kompromisseja ohjelmiston arkkitehtuurin noudattamisessa sekä tehdä muutoksia ymmärtämättä täysin niiden vaikutusta muun järjestelmän toimintaan [Eick *et al.* 2001]. Kehittäjä ottaa tällöin niin sanottua *teknistä velkaa* (technical debt), joka tarkoittaa käytännössä sellaisten teknisten ratkaisujen rakentamista, jotka tulevat tarvitsemaan jatkossa lisää huomiota, sillä nämä tehdyt ratkaisut ovat laadultaan huonoja ja ne on tehty lähinnä lyhyen tähtäimen hyödyn saavuttamiseksi [Rocha *et al.* 2017]. Tekninen velka voi olla tahallista, jolloin ohjelmiston kehityksessä ei esimerkiksi priorisoida tarvittavia korjauksia tai muutoksia uusien toimintojen lisäyksien sijaan. Tahatonta tekninen velka on silloin, kun kehittäjät tekevät tietämättään teknistä velkaa kerryttäviä ratkaisuja. [Tunttunen 2014]

Teknistä velkaa voidaan verrata taloudelliseen velkaan; tekninen velka mahdollistaa nopean kehityksen myöhemmin maksettavan lainan koron kustannuksella [Buschmann 2011]. Teknisellä velalla voidaan siis kuvata monia erilaisia laatuongelmia, joihin tulee reagoida jossakin vaiheessa tulevaisuudessa [Rocha *et al.* 2017]. Jotta teknistä velkaa ei kertyisi on kaikissa tilanteissa erittäin keskeistä pitää kiinni ohjelmiston tavoitteellisesta laatutasosta: tällöin on keskeistä pyrkiä huomioimaan ohjelmiston arkkitehtuuri ja mahdollisimman yhtenäinen ohjelmakoodi kaikissa tilanteissa [Rocha *et al.* 2017].

Tekninen velka ei kerry vain ohjelmistoa kehittävien tahojen toimesta, vaan järjestelmään jo rakennetut toiminnot voivat muuttua myös ajan saatossa tekniseksi velaksi, vaikka ne olisi rakennettu erittäin hyvin järjestelmän arkkitehtuuria kunnioittaen [Rocha *et al.* 2017]. Teknistä velkaa voi syntyä myös muista ulkoisista syistä. Esimerkiksi

PHP:lle on tyypillistä ohjelmakoodin rakenteen muutos eri versioiden välissä ja muutokset esimerkiksi funktioissa ja tyypityksessä [Cholakov 2008]. Tämä voi aiheuttaa teknistä velkaa synnyttämällä tarpeen muokata kaikki ohjelmiston osaset käyttämään jotakin uudessa versiossa tullutta toimintoa, jonka myötä jokin vanha toiminto saattaa esimerkiksi poistua tulevissa versioissa.

Ohjelmiston rappeutumista voidaan estää kehittämällä sellaisia ohjelmiston kehitystapoja, joissa laadun tarkkailu on koko ajan läsnä [Silva ja Balasubramaniam 2012]. Näitä kehitystapoja avataan tarkemmin alaluvussa 3.3, jossa käsitellään laadun parantamista. Parempien kehitystapojen lisäksi on myös järkevää pyrkiä varaamaan kehitystyölle sen vaatima aika, jotta kehittäjät eivät tuottaisi laadultaan heikkoja ratkaisuja kovan aikataulupaineen alla [Eick *et al.* 2001]. Aikataulupaineen vuoksi myös tiedon siirtäminen projektista tai kehitystyöstä toiseen voi olla erittäin hankalaa [Lyytinen ja Robey 1999]. Tällöin esimerkiksi uusista ohjelmiston arkkitehtuuriin liittyvistä piirteistä syntyvä tieto ei välttämättä kulje eri kehittäjien välillä ja ohjelmiston laatu heikkenee. Austin [2001] suosittaakin tutkimuksessaan, että työarvioiden ja aikataulujen arvioinnissa käytettäisiin myös systemaattisesti hyväksi aiempien kehitystöiden työarvioiden paikkansapitävyyden historiatietoa. Tutkimuksessa painotetaan kuitenkin myös sitä, että ei ole järkevää asettaa sellaisia aikatauluarvioita, jotka pystytään aina alittamaan, sillä tällöin työn tuottavuus saattaa tietoisesti laskea [Austin 2001].

Ohjelmiston rappeutumisen estäminen ja hallinta ovat erittäin tärkeitä elementtejä ohjelmiston laadun näkökulmasta ja mikäli niihin ei kiinnitetä huomiota, nousevat myös järjestelmän kehityskulut vähitellen sen elinkaaren aikana [Rocha *et al.* 2017]. Web-pohjaisissa järjestelmissä kehittäjien tulee yleensä osata erittäin monipuolisesti eri tekniikoita ja aikataulupaineet ovat kovia [Komara *et al.* 2016]. Tällöin moni aiemmin esitellyistä ohjelmiston rappeutumista aiheuttavista riskitekijöistä realisoituu ja myös teknistä velkaa voi syntyä. Ehkä tästäkin syystä web-pohjaisten järjestelmien maine laatutason osalta ei ole kovin imarteleva [Nederlof *et al.* 2014].

Lehmanin lakien mukaisesti ohjelmisto tulee aina muuttumaan jatkuvasti koko elinkaarensa aikana uusien ja muuttuvien vaatimusten sekä erilaisten korjausten ja muokkausten myötä [Sommerville 2007]. Jotta ohjelmisto säilyisi koko elinkaarensa aikana halutulla laatutasolla, tulisi laatutasoa seurata aktiivisesti ja välttää esimerkiksi teknisen velan tahatonta ja tahallista kumuloitumista sekä sellaisia prosesseja, jotka edistävät ohjelmiston rappeutumista.

Ohjelmiston laatutasoa ja sen kehittymistä voi parhaiten seurata erilaisten laatutason seuraamiseksi määriteltyjen mittaristojen kautta. Esimerkiksi mitatun suorituskyvyn lasku voi olla selkeä laatuongelmien oire [Alcocer ja Bergel 2015]. Toisaalta taas hyvin nopea ohjelmisto ei välttämättä ole muokattavissa ollenkaan ja hajoaa heti tehtäessä pieninkin muutos ohjelmakoodiin, jolloin ohjelmiston ei voida sanoa

olevan kovin laadukas [Silva ja Balasubramaniam 2012]. Näin ollen onkin keskeistä, että ohjelmiston laatutasoa seurataan sopivilla mittaristoilla, jotka kertovat olennaisen juuri tämän ohjelmiston tilanteesta. Kun mittarit ovat kunnossa, voidaan ohjelmiston laatutasoa seurata objektiivisesti koko ohjelmiston elinkaaren ajan ja pureutua myös laadun muutoksien taustalla oleviin syihin.

3.2 Laadun arviointi ja metriikat

Jotta laatua voitaisiin arvioida ja sen kehitystä mitata, on määriteltävä milloin ohjelmistoa voidaan pitää laadukkaana. Suurempien kehitysorganisaatioiden, ohjelmistojen ja tietojärjestelmien tapauksessa tätä varten voidaan tehdä oma erillinen laatusuunnitelma osana laadun hallintaa laadunvarmistuksen parissa. Kaiken kaikkiaan keskeisintä on kuitenkin, että kaikki järjestelmää kehittävät ja sen parissa työskentelevät tahot ovat selvillä siitä, mitä hyvä laatu juuri tämän järjestelmän tapauksessa tarkoittaa. Ilman tätä määritelmää erilaisia päätelmiä ja olettamuksia hyvästä laadusta ja sen muodosta voidaan tehdä useita, mikä altistaa järjestelmän kehityksen alttiiksi ristiriitaisille tavoitteille. [Sommerville 2007]

Kun hyvän laadun taso on määritelty, voidaan määritellä sen toteutumisen arvioimista varten sopivat arviointitavat ja metriikat. Yleisellä tasolla metriikat voivat olla mitä tahansa mittareita, jotka liittyvät käsillä olevaan järjestelmään. Metriikoita voidaan käyttää yleisesti kahdenlaisten arvioiden johtamiseen järjestelmän tilasta: yleiset arviot koko järjestelmästä ja erilaisten anomalioiden tunnistaminen järjestelmässä. Yleisten arvioiden teolla tarkoitetaan prosessia, jossa johdetaan tietynlaisten mittareiden tuloksista uutta tietoa. [Sommerville 2007] Tämä voi tarkoittaa esimerkiksi web CMS -järjestelmien tapauksessa järjestelmän kompleksisuuden arviointia yksittäisten kutsujen suoritusaikojen ja SQL-kyselyjen lukumäärän perusteella. Järjestelmien anomalioiden tunnistamisella taas tarkoitetaan prosessia, jossa tunnistetaan järjestelmän eri osien toiminnasta anomalioita metriikoiden avulla [Sommerville 2007]. Tällöin edellistä esimerkkiä soveltaen yhden järjestelmän osan suoritusajan hidastuminen ja SQL-kyselyjen määrän kasvu voivat kieliä tässä järjestelmän osassa olevasta ongelmasta.

Ei ole olemassa yleisiä ja kaikkiin erilaisiin sovelluskohtiin sopivia metriikoita [Sommerville 2007]. Metriikoiden asettamisessa onkin keskeistä, että ne heijastavat järjestelmältä odotettuja asioita. Tällöin on tärkeää pohtia erityisesti seuraavia asioita valittaessa sopivia metriikoita; Ensimmäisenä keskeisenä asiana tulisi pohtia kuka on metriikoiden asiakas, eli minkälaisia asioita metriikoiden tulisi heijastaa ja mitkä asiat metriikoiden tuloksista korostuvat. Huonosti valitut metriikat voivat heijastaa vääränlaisia tavoitteita ja ohjata muokkaamaan järjestelmää väärillä tavoilla. Toinen keskeinen pohdittava asia onkin, mitä ovat näiden asiakkaiden tavoitteet järjestelmän mittauksessa suhteessa tuotteeseen, prosessiin ja resursseihin. Kolmas keskeinen pohdittava asia on, miten metriikoiden tuottamien tulosten avulla pystytään osoittamaan,

että laadulle asetetut tavoitteet on pystytty täyttämään tai miten tilanne on ylipäätään muuttunut. [Laird ja Brennan 2006]

Metriikoiden tulisi olla monipuolisia, ja niiden toimintaa on hyvä tarkastella kriittisesti aina aika ajoin, sillä metriikoille on tyypillistä, että ne voivat menettää tehokkuutensa vähitellen ajan myötä [Laird ja Brennan 2006]. Tällöin liian yksinkertaisten tai yksipuolisten metriikoiden valinta voi johtaa siihen, etteivät metriikat todellisuudessa heijasta järjestelmän tilaa ja huomio siirtyy seikkoihin, jotka saattavat jopa heikentää järjestelmän laatutavoitteisiin pääsyä [Laird ja Brennan 2006]. Esimerkiksi pelkästään web CMS -järjestelmän suoritusaikoihin kohdistuva mittaaminen voi johtaa siihen, että palvelimen kuorma nousee merkittävästi ja kantokyky useiden yhtäaikaisten kutsujen yhteydessä heikkenee.

Kun käytettävät metriikat on saatu valittua, olisi hyvä luoda jokin ympäristö, jossa metriikoiden kehitystä ja tasoa voidaan seurata. Tämän ympäristön avulla voidaan käyttää *vakioituja mallintavia testejä* (benchmark) mittauksien suorittamiseen. Nämä vakioidut testit suunnitellaan sellaisiksi, että ne vastaavat mahdollisimman hyvin normaaleja järjestelmän käyttötilanteita ja antavat siten mahdollisimman tarkan tuloksen kohteeksi valitun metriikan kehityksestä. Vakioitujen testien tarkoitus on aina tuottaa sellaista dataa, jonka perusteella voidaan päättää, miten järjestelmää viedään eteenpäin ja miten metriikan tulokset ovat kehittyneet suhteessa vanhoihin aiemmin suoritettuihin testeihin. [Laird ja Brennan 2006] Vakioitujen testien ajamiseen voidaan käyttää myös erityisesti testausta varten kehitettyjä erilaisia järjestelmiä, automaatiota ja esimerkiksi yksikkötestejä, mutta näihin ei paneuduta tässä tutkielmassa.

Tässä tutkielmassa käsitellään laatua lähinnä suorituskyvyn näkökulmasta ja seuraavassa tämän alaluvun osassa esitelläänkin joitakin suorituskykyyn liittyviä käytettävissä olevia metriikoita, joita voitaisiin soveltaa myös PHP-pohjaisissa web CMS -järjestelmissä. Järjestelmien yleisen laatutason seurantaan suositetaan useissa eri julkaisuissa käytettäväksi versionhallinnan avulla tehtävää jonkintasoista erilaisten muutosten seurantaa [Rocha *et al.* 2017, Laird ja Brennan 2006, Silva ja Balasubramaniam 2012]. Seuraamalla versionhallintaa voidaan sen kautta analysoida järjestelmän laatutasoa arvioimalla esimerkiksi virheiden esiintyvyyden ja sijainnin määrän muutoksia [Laird ja Brennan 2006]. Muutosten arviointi onkin yksi keskeinen osa yleistä laadunvarmistusprosessia ja suuremmissa organisaatioissa on yleensä oma henkilöstönsä nimettynä juuri laadunvarmistusta varten [Sommerville 2007].

PHP-pohjaisissa web CMS -järjestelmissä voidaan käyttää yleisinä metriikoina muun muassa suoritusaikaa, SQL-kyselyjen lukumäärää sekä palvelimen *keskusmuistin* (RAM) ja *prosessorin* (CPU) käytön tietoja. Nämä metriikat ovat hyviä PHP:n tapauksessa, sillä niiden huonot arvot kielivät yleensä ongelmista jossain osassa järjestelmää, johon myös Arsenault [2017] viittaa. Suoritusajan kasvu kielii yleensä

jostain muusta ongelmasta järjestelmän sisällä. Esimerkiksi SQL-kyselyjen suuri määrä nostaa helposti suoritusajan hyvin suureksi [Arsenault 2017].

Web CMS -järjestelmien suorituskyvyn mittaamiseen voidaan käyttää myös ohjelmakoodin analysointiin liittyviä työkaluja. Staattisen analyysin avulla voidaan tulkita ohjelmakoodia sitä suorittamatta ja etsiä siitä vikoja ja esimerkiksi tehokkuusongelmia [Hauzar ja Kofroň 2014]. PHP:n tapauksessa kielen dynaamisuus aiheuttaa staattisen analyysin käytölle kuitenkin ongelmia, minkä takia se ei ole kovin helppo tapa ohjelmakoodin analysoinnissa [Hauzar ja Kofroň 2014, Hills *et al.* 2014]

PHP:n tapauksessa voidaan käyttää ohjelmakoodin profilointia Xdebug PHP-laajennoksen avulla. Ohjelmakoodin profilointi kerää käyttäjän tekemästä kutsusta profiilin, josta selviää muun muassa kunkin funktion suoritusaika ja kutsujen lukumäärä sekä se, mistä kutsut ovat tulleet ja mitä mistäkin funktiosta kutsutaan. Profiloinnin avulla saadaan myös kerättyä tieto keskusmuistin käytöstä eri funktioissa. [Xdebug 2019] Koko järjestelmän profilointi on aikaa vievä prosessi, mutta lopputulemana saadaan erittäin tarkka kuva järjestelmän suorituskyvyn tilasta [Arsenault 2017].

Kun sopiva laatutaso, mittaristot, vakioidut testit ja mittaristojen keräämän datan arviointitavat on saatu määriteltyä, tulisi nämä tallentaa kirjallisessa muodossa talteen myöhemmin käytettäväksi. Nämä organisaatiolle luodut laatutason arvioimiseksi käytettävät työkalut muodostavat organisaatiolle sen oman laatujärjestelmän pohjaversion. [Sommerville 2007] On olemassa myös valtava määrä erilaisia standardoituja laatujärjestelmiä, joita voidaan käyttää ohjelmiston kehityksessä halutun laatutason varmistamiseksi.

Laatujärjestelmät ja standardit tarjoavat useita erilaisia hyötyjä ohjelmistojen kehityksessä. Niiden yleinen tavoite on toimia osana ohjelmistojen kehityksen laadunhallintaa ja nostaa ohjelmistojen sekä niiden kehitykseen käytettyjen prosessien laatua yhteisten ennalta määriteltyjen mallien, sääntöjen, toimintatapojen ja prosessien kautta [Sommerville 2007].

Laatujärjestelmien ja standardien keskeisin hyöty syntyy tiedon jakamisen ja ymmärryksen yhtenäistämisen kautta, sillä standardit ja laatujärjestelmät pyrkivät poistamaan mahdollisuuden eri ihmisten välisiin asioiden subjektiivisiin tulkintaeroihin [Schneidewind 1996]. Standardeihin ja laatujärjestelmiin tallentaan yleensä jostakin näkökulmasta parhaaksi lähestymistavaksi uskottu toimintatapa, jolloin ne auttavat huomioimaan ohjelmiston laadun osana ohjelmiston kehitystä [Sommerville 2007].

Laatujärjestelmät ja standardit voivat myös antaa jonkinlaisen viitekehyksen, jonka varaan esimerkiksi koko laadunhallinnan prosessi voidaan rakentaa organisaation sisällä. Laadunhallinnan yhtenä yleisenä tavoitteena onkin varmistaa, että jonkin ohjelmiston kehitykseen valitut standardit on myös otettu käyttöön ja että niitä käytetään

jatkuvasti osana ohjelmiston kehitystä koko ohjelmistotuotantolinjassa. [Sommerville 2007]

Kun ohjelmiston rakenne ja prosessit on muotoiltu käyttäen jotakin järkevästi muotoiltua standardia, on myös helpompi tehdä vaihdoksia esimerkiksi jotakin toiminnallisuuskokonaisuutta toteuttavien kehittäjien välillä [Sommerville 2007]. Tällöin myös eri kehittäjien tuottaman ohjelmakoodin laatu ja ylläpidettävyys kasvavat, kun tuotettu ohjelmisto on toteutettu käyttäen yhteisesti käytettäviä menetelmiä ja rakenteita [Schneidewind 1996]. Kehittäjät mainitsevat hyvien käytäntöjen noudattamisen yhtenä kehittäjälle kuuluvana vastuualueena [Rocha *et al.* 2017]. Standardien avulla kehittäjälle jää kuitenkin enemmän aikaa itse ratkaisujen toteuttamiseen parhaiden käytäntöjen ollessa jo määriteltyinä standardien ja laatujärjestelmien sisälle.

Standardien ja laatujärjestelmien koko ja niiden kattamat erilaiset asiat vaihtelevat paljon. ISO 9000 koostuu monista erilaisista laadun hallintaan ja kehittämiseen liittyvistä asioista, jotka eivät kuitenkaan rajoita esimerkiksi tapaa, jolla prosesseja tuotetaan, vaan asettaa lähinnä raamit sille, kuinka hyvin joitain luotuja prosesseja on noudatettava [Sommerville 2007]. Tällöin standardia käyttävälle organisaatiolle jää pitkälti vastuu sopivan toimintamallin muotoilusta. Suuria ja kattavia standardeja on kuitenkin myös kritisoitu niiden turhan massiivisten rakenteiden vuoksi [Fenton 1996]. Esimerkiksi ISO 25030 antaa taas vain viitekehyksen sille, miten ohjelmiston vaatimuksia tulisi muotoilla [Bøegh 2008]. Tällöin organisaatiolle itselleen ei jää kovin paljoa väljyyttä standardin toteutustavan osalta.

Laatujärjestelmien ja standardien käyttäminen osana ohjelmiston kehitystä on keskeistä, sillä tällöin pystytään pitämään paremmin huolta esimerkiksi ohjelmiston rappeutumisen kehityksestä vaikuttamalla suoraan hyväksyttyyn ohjelmiston tasoon [Sommerville 2007]. Joskus tilanne voi kuitenkin olla se, etteivät ohjelmiston tekninen taso ja laatu vain ole hyväksytyllä tasolla ja on pakko tehdä suuriakin muutoksia tason nostamiseksi.

3.3 Laadun parantaminen

Kun ohjelmiston arvioimiseksi rakennetut mittarit ja testiympäristöt on saatu muotoiltua sekä mittaukset tehtyä, voidaan siirtyä laadun parantamiseen saatujen tulosten perusteella. Laadun parantamisessa pyritään tulkitsemaan mittareiden tuottamia tuloksia ja tunnistamaan niistä huomiota vaativia ongelmakohtia, joita muokkaamalla ohjelmiston todellista laatutasoa saadaan nostettua lähemmäs kohti tavoitetasoa. Laadun parantamiseksi voidaan tehdä hyvin paljon erilaisia toimenpiteitä, jotka voivat kohdistua esimerkiksi ohjelmiston tekniseen puoleen tai kehitysorganisaation käyttämiin prosesseihin. [Sommerville 2007] Tässä alaluvussa keskitytään prosessien sijaan yleisiin menetelmiin, joilla huonolaatuista ohjelmakoodia voidaan parantaa, sekä erityisesti PHP-

pohjaisissa web CMS -järjestelmissä käyttökelpoisiin laadun parantamisen toimenpiteisiin.

Mikäli ohjelmistoon on kertynyt paljon teknistä velkaa ja tämä on muodostunut ohjelmiston rappeutumisen myötä ohjelmiston laatuun merkittävästi vaikuttavaksi tekijäksi, tulee pohtia tulisiko tekninen velka maksaa jollakin tavalla pois sen sijaan, että yritetään tulla toimeen huonon laadun kanssa [Rocha *et al.* 2017]. Tekninen velka voidaan myös muokata erilaiseen muotoon tekemällä esimerkiksi väliaikaisia ratkaisuja. Jos tekninen velka halutaan kuitenkin maksaa pois, on sen maksamiseksi kolme erilaista pääkeinoa: ohjelmakoodin *refaktorointi* (refactoring), uudelleen suunnittelu sekä uudelleen kirjoittaminen. Valittavan lähestymistavan tulisi myös riippua siitä, mikä on käytettävissä olevista ratkaisuvaihtoehdoista taloudellisesti mielekkäin. [Buschmann 2011] Onkin tärkeää, että kehitysorganisaatiossa osataan arvioida sekä teknisen velan taloudellisia että teknisiä piirteitä tasapuolisesti [Tunttunen 2014]. Tällöin pystytään valitsemaan kokonaisuuden kannalta paras vaihtoehto laadun parantamiseksi.

Ohjelmiston osan uudelleen suunnittelu ja uudelleen kirjoittaminen tarkoittavat käytännössä ohjelmakoodin tarkastelua ja sen kirjoittamista uuteen yhtenäisempään sekä selkeämpään muotoon. Nämä toimenpiteet ovat työläitä, eikä niiden käyttämisessä ole välttämättä edes tavoitteena säilyttää aiemmin järjestelmän tukemia erilaisia käyttötapauksia. [Laguna ja Crespo 2013] Tällöin uudelleen suunnittelu ja kirjoittaminen eivät ole välttämättä sopivia vaihtoehtoja, vaan refaktorointi voisi toimia paremmin.

Refaktoroinnin tavoitteena on muokata ohjelmistoa siten, ettei sen ulos heijastuva toimintalogiikka muutu sen sisäisen rakenteen parantuessa. Refaktorointia voidaankin kuvata ikään kuin ohjelmiston rappeutumisen vastakohdaksi; sen avulla parannetaan ohjelmiston rakenteita niiden hitaan rappeutumisen sijaan. Refaktorointia suositetaankin tehtäväksi oikeastaan koko ajan pienissä erissä muun ohjelmistokehityksen rinnalla, jolloin ohjelmiston rappeutumista vastaan tehdään koko ajan töitä ja siten parannetaan myös laatua. Refaktorointi aktiivisena osana ohjelmiston kehitystä nopeuttaa myös ohjelmiston kehitystyötä vähentämällä ohjelmiston rappeutumisen myötä kumuloituvia laatuongelmia. [Fowler *et al.* 1999]

Refaktorointia on hyvä tehdä aina, kun ohjelmakoodin huomataan muuttuvan sen alkuperäisestä tarkoituksesta yhä enemmän kohti epämääräistä ja hankalasti seurattavaa ohjelmakoodia. Mikäli jokin osa ohjelmakoodissa tuntuu intuitiivisesti muuta järjestelmää vastaan taistelevalta, tulisi se muokata sopivaksi. [Fowler *et al.* 1999] Olio-ohjelmointiin pohjautuvissa järjestelmissä on pidetty hyvänä tapana suorittaa refaktorointia perustuen järjestelmässä yleisesti hyväksyttyjen ja ennalta määriteltyjen ohjelmiston rakenteiden ja muotoseikkojen kuvauksiin [Neill ja Laplante 2006]. Kun nämä ohjelmiston rakenteiden toteutustavat ja eri muotoseikat on kuvattu ja dokumentoitu, on myös otettu jo ensimmäiset askeleet kohti ohjelmiston rakenteen

laadun seurantaa pohjautuen jonkinlaiseen sisäisesti hyväksyttyyn laatustandardiin. Ei ole olemassa yksittäistä kaikille erilaisille kehittäjille ja ohjelmistoille soveltuvaa syntaksia [Santos ja Gerosa 2018]. On siis järkevää pyrkiä muodostamaan ja dokumentoimaan juuri jossakin tietyssä kontekstissa parhaiten toimivat käytännöt.

Refaktoroinnille voidaan myös asettaa tavoitteita, esimerkiksi suorituskyvyn osalta refaktoroinnin tulisi aina parantaa tai säilyttää suorituskyvyn nykyinen taso [Mens ja Tourwé 2004]. Suorituskyvylle on kuitenkin tyypillistä, että sen taso vaihtelee hieman [Alcocer ja Bergel 2015]. Näin ollen refaktoroinnissa ei tulisi suhtautua esimerkiksi suorituskykyyn täysin absoluuttisesti, sillä ohjelmakoodin seurattavuus ja teknisen velan vähentäminen ovat yleensä tärkeämmässä roolissa.

Refaktoroinnin toimia voidaan myös ohjata esimerkiksi sellaisiin kohteisiin, joissa olio-ohjelmointiin perustuvissa järjestelmissä on muotoutunut samankaltaisia luokkia, joita kaikkia joudutaan kuitenkin ylläpitämään. Tällöin näitä voidaan eriyttää yksittäisiksi yläluokiksi ja yläluokan periviksi alaluokiksi, jolloin abstraktion määrän kasvun myötä myös ohjelmiston laatua voidaan parantaa ja toisaalta ylläpidettävän ohjelmakoodin määrää laskea [Tsantalis ja Chatzigeorgiou 2010].

Refaktorointi sopii hyvin ohjelmiston rakenteellisten laatuongelmien ratkaisemiseen, mutta sen lisäksi pystytään käyttämään myös muita keinoja laadun parantamiseksi suorituskyvyn näkökulmasta. PHP-pohjaisissa järjestelmissä tietokannan käyttöön voi kulua erittäin merkittävä osa suoritusajasta [Arsenault 2017]. Tällöin voidaan kuitenkin käyttää esimerkiksi erilaisia välimuisteja suorituskyvyn nostamiseksi sekä tietokannan kuormituksen vähentämiseksi [Sa'adah *et al.* 2015].

Ohjelmointikielenä PHP ei ohjaa itsessään käyttämään parhaita mahdollisia ratkaisuja, vaan sen parissa on mahdollista kehittää hyvin paljon erilaisia toimintatapoja ja ohjelmointityylejä [Cholakov 2008]. Tällöin PHP-pohjaisissa järjestelmissä on erittäin tärkeää pyrkiä ottamaan huomioon suorituskyvyn näkökulmasta parhaat mahdolliset toteutustavat, joilla ohjelmakoodin suoritusta pystytään optimoimaan merkittävästi nopeammaksi ilman suuria vaikutuksia esimerkiksi järjestelmän skaalautuvuuteen [Mohammad 2017].

Erinäisiä listauksia hyvistä PHP-koodin optimointikeinoista ohjelmakoodin tasolla on Internetissä paljon [Mohammad 2017, Arsenault 2017, Bradley 2018]. Kun näitä erilaisia keinoja kokeillaan, olisi jossakin järjestelmässä parhaiksi ja suorituskykyisimmiksi havaittuja keinoja hyvä dokumentoida järjestelmän kehityksessä yleisesti käytetyiksi laatuohjeiksi esimerkiksi osana jonkinlaista laatujärjestelmää, jotta niiden avulla saatava suorituskyvyn parannus säilyy pitkäjänteisenä osana ohjelmiston kehitystä [Sommerville 2007]. Tällöin koko ohjelmiston arkkitehtuurin kehitystä voidaan ohjata rakennettujen laadullisten ohjeistuksien avulla [Silva ja Balasubramaniam 2012].

PHP-pohjaisissa järjestelmissä on myös erittäin tärkeää pysyä aina ohjelmakoodin versioiden kehityksessä mukana, sillä uusimmissa PHP:n versioissa on yleensä aina suorituskyvyn kannalta merkittäviä parannuksia [Mohammad 2017]. Tästä kertovat myös eri PHP-versioilla tehdyt suorituskyvyn testit. Uusimmat PHP:n versiot ovat lähes poikkeuksetta useissa erilaisissa testeissä kaikista suorituskykyisimpiä jopa useiden kymmenien prosenttien marginaaleilla [Gavalda 2019]. Laadun parantamiseksi olisikin siis hyvä käyttää aina uusinta ohjelmakoodin versiota, vaikka PHP:tä onkin kritisoitu siitä, että PHP:n kehitystiimi tekee kohtalaisen suuria muutoksia jokaisessa julkaisuversiossa [Cholakov 2008].

Ohjelmistoja voidaan parantaa myös staattisen analyysin ja optimoinnin avulla. Tämä ei kuitenkaan ole välttämättä aina mahdollista ja esimerkiksi PHP:n osalta kielelle tyypillinen dynaamisuus hankaloittaa staattisen optimoinnin käyttöä. Lisäksi täysin staattinen optimointi voi kasvattaa ohjelmakoodin määrää merkittävästi. [Popov *et al.* 2017] Ohjelmisto voi olla suorituskykyinen olematta laadukas [Silva ja Balasubramaniam 2012]. Staattisen optimoinnin tuloksena ei tällöin ole välttämättä laadullisesti parantunut ohjelmisto, vaikka sen suorituskyky olisikin parantunut.

Staattisen optimoinnin tavoin PHP-koodin esikääntämistä voidaan käyttää myös suorituskyvyn nostamiseen [Alieksieiev ja Bondarenko 2012]. Esikääntämisessä on kuitenkin samankaltainen ongelma kuin staattisessa optimoinnissa: vaikka se vaikuttaisi nostavan suorituskykyä, se ei kuitenkaan pureudu pinnan alla piileviin suurempiin laadullisiin ongelmiin. Suorituskyvyn lasku voi olla oire laatuongelmista [Laird ja Brennan 2006], jolloin näihin pureutuminen osana suorituskyvyn nostoa on oleellista. Seuraavassa luvussa 4 käsitellään tämän tutkielman tapaustutkimuksen kohteena olevaa web CMS -järjestelmää, jonka IT-alalla toimiva yritys on kehittänyt. Web CMS -järjestelmään määritellään suorituskyvyn mittaukseen soveltuvat mittaristot, testiympäristö ja vakioitu testidata.

4 Tapaustutkimuksen web CMS -järjestelmä

Tämän tutkielman tapaustutkimuksen kohteena on IT-alalla toimivan yrityksen kehittämä web CMS -järjestelmä ja sen versio 19.0. Järjestelmän yhteyteen on kehitetty myös keskeisenä osana verkkokauppajärjestelmä, joka onkin oikeastaan järjestelmän pääpainopisteenä. Sisällönhallinta on lähinnä verkkokauppajärjestelmän toimintaa tukeva toiminnallisuuksien kokonaisuus, joskin erittäin keskeinen sellainen. Järjestelmästä puhuttaessa viitataankin sekä sisällönhallinta- että verkkokauppaosiin, jotka muodostavat yhdessä tämän web CMS -järjestelmän. Web CMS -järjestelmä on yksi yrityksen tarjoamista palveluista, ja se on integroitu kokonaan yrityksen tarjoamiin muihin ratkaisuihin erityisesti verkkokaupan toiminnallisuuksien osalta.

Tämä web CMS -järjestelmä sisältää erikseen julkisen puolen sekä ylläpidon käyttöön rajatun hallintapaneelin. Hallintapaneelin kautta järjestelmässä voi rakentaa paljon erilaisia sisältöjä: sivuja, uutisia, tiedotteita, bannereita, kalentereita ja muun muassa erilaisia mediakirjastoja. Järjestelmässä pystyy luomaan eritasoisia käyttäjäryhmiä ja rajaamaan sisältöjen näkyvyyksiä esimerkiksi käyttäjäryhmien perusteella sekä erilaisten ajanjaksojen kautta. Sisällöille voi luoda erilaisia hierarkioita, ja järjestelmän yleistä toimintaa pystyy hallitsemaan monipuolisten asetuksien kautta.

Järjestelmän sisällöistä pystyy luomaan responsiivisia hallintapaneelin WYSIWYG-editorien avulla, mutta sisältöjä ei pysty tuomaan järjestelmään suoraan jostakin ulkoisesta palvelusta tai esimerkiksi .csv-tiedostoista. Järjestelmä on kielellistetty, ja sen avulla pystyy myös erottelemaan erilaisia sisältöjä eri kieliryhmille ja maille. Sisältöjen pohjalla toimii joukko staattisia ja pääosin järjestelmää kehittävien tahojen ylläpitämiä HTML-sivuja, joihin sisältö muodostetaan dynaamisesti kutsujen yhteydessä web CMS -järjestelmän tietokantaan ja tiedostojenhallintaan tallennettujen tietojen ja asetuksien perusteella.

Sisällönhallintapuolen tavoin myös verkkokaupassa näkyviä tuote- ja tuoteryhmäsisältöjä pystytään hallinnoimaan hallintapaneelin kautta, vaikka verkkokaupan tuotteita ja tuoteryhmiä kontrolloidaankin pääasiassa siihen integroitujen ERP-järjestelmien kautta. Web CMS -järjestelmän näkyvyyksien hallinnan avulla voidaan myös luoda monipuolisia erilaisia tuote- ja sisältönäkyvyyksiä riippuen halutusta verkkokaupan asiakkaan asiakas- ja käyttäjäryhmätasosta. Sisällönhallintatoimintojen kautta voidaan myös luoda tavallista tuoteryhmärakennetta monipuolisempia erilaisia sisällöstä ja tuotenostoista koostuvia kokonaisuuksia.

Edellä kuvattujen toimintojen perusteella tapaustutkimuksen kohteena oleva web CMS -järjestelmä on Boikon [2001, 2005] kuvaamien erilaisten web CMS -järjestelmien tasojen perusteella täysimääräinen web CMS -järjestelmä, vaikkakaan järjestelmään ei voi tuoda nykyisellään sisällöksi muokattavaa tietoa ulkopuolisista järjestelmistä verkkokauppaa lukuun ottamatta.

Web CMS -järjestelmä on hieman yli kymmenen vuotta vanha, ja sen parissa alun perin työskennelleet kehittäjät eivät enää ole olleet useaan vuoteen yrityksen palveluksessa. Näin ollen järjestelmän arkkitehtuuri on vähitellen muotoutunut erilaisista kerroksista pääosin samankaltaista, mutta paikoitellen hyvinkin vaihtelevaa ohjelmakoodia. Tämä on varsin tyypillistä vanhemmille järjestelmille, jotka ovat muotoutuneet vähitellen ajan saatossa erilaisten paikallisten versiointien ja variaatioiden kautta [Laguna ja Crespo 2013]. Tällöin alkuperäinen järjestelmään suunniteltu arkkitehtuuri voi vähitellen muuttua sitä nykyisin kehittäville kehittäjille erittäin hankalaksi noudattaa ja toisaalta hahmottaa [Griffith *et al.* 2011].

Yrityksen web CMS -järjestelmän tuotekehitys on ollut sen elinkaaren aikana varsin asiakasprojektipainotteista, jolloin suurin osa uusista ominaisuuksista on rakennettu järjestelmään jonkin asiakasprojektin yhteydessä ja usein varsin tiukalla aikataululla. Järjestelmän parissa työskentelevät kehittäjät ovat olleet tiedoiltaan, taidoiltaan ja järjestelmän tuntemukseltaan vaihtelevia, jolloin järjestelmään rakennetut uudet ratkaisut eivät välttämättä ole aina olleet laadultaan niin tasokkaita kuin olisi toivottavaa. Tämä on aiheuttanut tahattoman teknisen velan kertymistä järjestelmään, ja tilanne onkin vastannut jossain määrin Rocha *et al.* [2017] kuvaamaa ohjelmistokehittäjien kokemusta teknisen velan syntymisestä.

Web CMS -järjestelmän tuotekehityksessä ei ole ollut käytössä erikseen noudatettavia laadunhallinnan kautta määriteltyjä järjestelmän arkkitehtuurin kehitystä ja rakennetta koskevia laatuohjeita. Vastuu järjestelmän laadun ylläpidosta on siis ollut hyvin pitkälti jotakin uutta toimintoa rakentavan kehittäjän käsissä, jolloin kehittäjien vaihteleva taso ja järjestelmän tuntemus on myös lisännyt alttiutta esimerkiksi teknisen velan syntymiselle sekä kokonaisarkkitehtuurin noudattamatta jättämiselle.

Web CMS -järjestelmän kokonaisarkkitehtuuri onkin asettunut erittäin voimakkaasti alttiiksi rappeutumiselle, sillä järjestelmän tuotekehityksessä on realisoitunut useita erilaisia rappeutumiseen johtavia riskejä, joita alaluvussa 3.1 on eritelty. Lisäksi suorituskyvyn näkökulmasta järjestelmän pitkäjänteinen suorituskyky ei yleisesti ole yleensä kovin korkealla prioriteetilla asiakasprojekteissa [Alcocer ja Bergel 2015]. Suorituskyvyn ongelmiin onkin vasta herätty yrityksessä, kun järjestelmän ongelmat ovat muuttuneet jo selkeästi joitakin suoritustilanteita haittaaviksi erityisesti monimutkaisempien sivukokonaisuuksien kohdalla.

Tapaustutkimuksen kohteena olevan web CMS -järjestelmän arkkitehtuuri noudattaa karkeasti *malli-näkymä-käsittelijä -arkkitehtuurin* (MVC) mukaista toteutustapaa [Cu *et al.* 2009] Siinä on järjestelmän ytimen muodostama kehys, joka suoritetaan jokaisen järjestelmään kohdistuvan kutsun yhteydessä. Järjestelmän yhteydet tietokantaan on eroteltu tarkasti muusta toiminnasta omiin malliluokkiinsa, mutta käsittelijä- ja näkymäosiot ovat osittain samoissa luokissa ja rakenteissa suoritettavia

toimintoja. Järjestelmän ohjelmakoodi sisältää myös paljon asiakaskohtaisten asetuksien kautta tehtyjä räätälöintejä järjestelmän oletustoimintojen seassa. Tämä on osaltaan aiheuttanut sen, että joidenkin luokkien toiminnallisuuden määrän kasvaessa ja asiakaskohtaisten räätälöintien lisääntyessä luokkien toiminnallisuuden seurattavuus on laskenut, ja siten arkkitehtuuri on päässyt rappeutumaan, kun tiukan aikataulun puitteissa ei ole välttämättä ehditty kiinnittämään tarpeeksi huomiota jo olemassa olevan arkkitehtuurin kehittämiseen ja ylläpitoon. Arkkitehtuurin rappeutumiselle onkin tyypillistä, että järjestelmän evoluution tapahtuessa hallitsemattomasti järjestelmä muuttuu vähitellen yhä monimutkaisemmaksi ja hankalammaksi ylläpitää [Hassaine *et al.* 2012].

Tapaustutkimuksen kohteena oleva web CMS -järjestelmä on PHP-pohjainen. Lisäksi järjestelmä käyttää MySQL-tietokantaa *PHP data objectin* (PDO) avulla ja sen ulkoasu on toteutettu HTML5:n ja SCSS:n avulla. JavaScript-tekniikoita on käytetty erilaisten toimintojen rakentamiseen ja AJAX-kutsuja soveltuvissa kohdin tietojen hakemiseen ilman sivujen latausta. Vaikka järjestelmän tuotekehitys onkin ollut pitkälti asiakasprojektipainotteista, on järjestelmän arkkitehtuuria myös parannettu aika ajoin merkittävien kehitystoimien myötä. Esimerkiksi tietokantakutsut olivat vielä taannoin osana järjestelmän muita luokkia, mutta nykyään ne on eroteltu täysin omiksi mallitason luokikseen ja tavallisimmat kyselyt on abstraktoitu omiksi funktiokutsuikseen.

Web CMS -järjestelmän PHP-puolen suorituskyky on kuitenkin muotoutunut viimeisen kahden vuoden aikana paikoitellen ongelmalliseksi tekijäksi järjestelmän kehityksessä ja uusien ominaisuuksien vaatimusten monimutkaistuessa. Sivujen latausajat ovat paikoitellen lähteneet venymään huomattavan suuriksi, mikä on varsin haitallista järjestelmän antaman laatuvaikutelman ja käyttökokemuksen kannalta [Laird ja Brennan 2006]. Suorituskyvyn ongelmien kumuloituessa vian etsintä on ollut myös hidasta ja kallista ilman selkeää prosessia tai yleisesti käytössä olevia ja hyväksi havaittuja sekä dokumentoituja työkaluja. Ongelmien etsintä onkin jäänyt helposti täysin arvailujen varaan sekä manuaalisesti käytettävien yksittäisten lievää hakuammuntaa olevien suorituskyvyn testien tekemiseen.

Suorituskyvyn seuranta ei ole ollut aktiivisena osana web CMS -järjestelmän tuotekehitystä ja versiointia esimerkiksi testausprosessin näkökulmasta. Näin ollen järjestelmän arkkitehtuurin kehityksessä ei ole aiemmin otettu järjestelmällisesti huomioon suorituskyvyn merkitystä ja rakennettujen ratkaisujen vaikutusta siihen. Tämän luvun alaluvuissa pyritään muotoilemaan tapaustutkimuksen kohteena olevalle web CMS -järjestelmälle suorituskyvyn mittaristo, jolla suorituskyvyn tilannetta pystytään mittamaan tarkasti ja pitkäjänteisesti. Lisäksi pyritään määrittelemään suorituskyvyn testauksessa käytettävä testiympäristö sekä testaukseen käytetyt testitapaukset.

4.1 Suorituskyvyn mittaristo

Tapaustutkimuksen kohteena olevan web CMS -järjestelmän suorituskyvyn mittaamiseen ei ollut ennalta määritettyä tai olemassa olevaa yleisesti käytettyä mittaristoa, joten käytettävä mittaristo oli etsittävä ja valittava tämän tutkielman yhteydessä. Vaatimukset mittaristolle asetettiin yhteistyössä yrityksen muiden ohjelmistokehittäjien kanssa. Järjestelmän suorituskyvyn ongelmien liittyessä lähinnä PHP-puolelle kohdistettiin mittaristot nimenomaan järjestelmän tähän osioon ja sen suorituskyvyn ongelmien analysointiin mahdollisimman tarkasti. Web CMS -järjestelmä asennetaan yleensä jossain määrin käytöltään rajattuun kohtalaisen yksinkertaiseen webhotelliin, mikä asetti valinnalle jonkin verran rajoitteita. Esimerkiksi pääosassa järjestelmän asennuksissa yleisimmin käytetyistä webhotelleista on erittäin hankala saada prosessorin ja keskusmuistin käytön tietoja järkevässä muodossa.

Suorituskyvyn mittariston tuli myös pystyä porautumaan yksittäisiin järjestelmän luokkiin ja funktioihin, jotta sillä pystyttäisiin tarvittaessa analysoimaan hyvin tarkasti erilaisia ohjelmakoodin mahdollisia ongelmakohtia sekä ylipäätään löytämään niitä valistuneiden arvauksien sijaan. Tämä on myös tärkeää, jotta pystytään arvioimaan web CMS -järjestelmän eri osioiden toimintaa kokonaisuudessaan ja arvioimaan järjestelmän kompleksisuutta ja arkkitehtuurin rakenteen toteutumista. Suorituskyvyn mittariston tuli näiden lisäksi antaa myös yleistä tietoa järjestelmän toiminnasta, eli yksittäisen suorituksen tarkkuudella tietoja suoritusajasta, SQL-kyselyjen lukumäärästä ja mahdollisuuksien mukaan keskusmuistin sekä suorittimen käyttöasteesta. Mittariston tuli olla myös sellainen, että se voitaisiin ottaa mahdollisimman vaivattomasti käyttöön tarvittaessa järjestelmän asennuskohteissa ilman järjestelmän toiminnan muuttumista ulospäin.

Edellisten vaatimusten myötä mittaristoiksi valikoitui kaksi erilaista suorituskyvyn mittaamiseen käytettävää mittaristoa. Ensimmäinen mittaristo valittiin porautuvuuden näkökulmasta, ja valinta kohdistui PHP:n laajennokseen Xdebugiin, joka on myös saatavilla suurimmassa osassa web CMS -järjestelmän tyypillisiä asennusympäristöjä. Tätä mittaristoa kutsutaan tutkielman tulevissa osissa porautuvaksi mittaristoksi. Xdebugilla pystytään profiloimaan ohjelmakoodia, jolloin profiilia analysoimalla esimerkiksi KCachegrind-sovelluksen avulla nähdään funktio- ja objektitasolle asti koko järjestelmän toiminta haluttujen kutsujen yhteydessä [Xdebug 2019]. Funktio- ja objektikutsuista nähdään myös muun muassa mistä niitä on kutsuttu, kuinka monta kertaa niitä on kutsuttu, mitä ne ovat kutsuneet ja kuinka paljon yksittäiset funktiokutsut ovat vieneet suoritusaikaa tai keskusmuistia. Xdebugin avulla suoritettava ohjelmakoodin profilointi onkin muotoutunut yhdeksi keskeiseksi tavaksi parantaa PHP-pohjaisten web-järjestelmien suorituskykyä, ja sitä suositellaankin tehtäväksi usein optimoinnista puhuttaessa [Arsenault 2017, Mohammad 2017].

Xdebug on myös erittäin helppo ottaa käyttöön, sillä se voidaan asettaa käytettäväksi php.ini-tiedostoon lisättävien PHP:n konfiguraatioasetusten avulla webhotelliympäristössä. Xdebugin profiloijan käyttö on myös rajattavissa php.ini-tiedostossa määritellyn konfiguraation avulla käytettäväksi vain tiettyjen kutsujen kohdalla. Tämä mahdollistaa sen, ettei Xdebug vaikuta juurikaan web CMS -järjestelmän normaaliin toimintaan, sillä profiloijan käyttö kutsujen yhteydessä hidastaa kyseisten kutsujen suoritusta erittäin merkittävästi. Profiloijan tuottamat profilointitiedostot ovat myös erittäin suuria. [Xdebug 2019] Tämän vuoksi profiloinnin suorittaminen vain tietyillä parametreillä varustettujen kutsujen yhteydessä on tärkeää webhotelleille tyypillisen pienen levytilan vuoksi.

Yleismittaristoksi tutkielmassa jatkossa kutsutuksi mittaristoksi valikoitui web CMS -järjestelmään erikseen tämän tutkielman yhteydessä rakennettava ja .csv-tiedostoon suoritusarvoja riveinä kutsujen yhteydessä tallentava mittaristo. Yleismittaristo kohdennettiin yksittäisen järjestelmän asetuksella määritellyn kirjautuneen käyttäjän suorittamien kutsujen yhteydessä toimivaksi, joten sen käyttö ei sinällään nosta järjestelmän kuormitusta juurikaan. Yleismittaristo on sisällytetty tapaustutkimuksen kohteena olevan web CMS -järjestelmän tuleviin versioihin, jolloin se on käytettävissä uusissa sekä päivitetyissä järjestelmissä suoraan ilman ylimääräisiä toimenpiteitä mittaristoille annettujen vaatimusten mukaisesti.

Yleismittaristo kerää ohjelmakoodin suorituksesta seuraavat tiedot: suoritusajan sekunteina, SQL-kyselyjen lukumäärän, kutsun kohdeosoitteen, suoritusajankohdan sekä tiedon PHP:lle keskimäärin varatusta keskusmuistin määrästä kutsun aikana. Sen avulla pystytään arvioimaan järjestelmän yleistä suoritustasoa siihen kohdistettujen kutsujen tuottamien tuloksien perusteella. Webhotellin suorittimen käyttötietoja tai täysin tarkkoja keskusmuistin käyttömääriä ei pystytty lisäämään mittaristoon mielekkäällä tavalla webhotellin asettamien rajoitusten vuoksi.

Mittaristojen tuottama data ei ole kuitenkaan mielekästä ja järjestelmän pitkäjänteisen kehittämisen näkökulmasta tarkkaa, mikäli datan tuottamiseksi käytetyn järjestelmän suoritusympäristö sekä erilaiset testitapaukset ja järjestelmän konfiguraatio vaihtelevat [Laird ja Brennan 2006]. Seuraavassa alaluvussa on eritelty järjestelmän testiympäristö sekä suorituskyvyn mittauksessa käytetyt erilaiset testitapaukset.

4.2 Käytetty testiympäristö, -data ja -tapaukset

Mitattavien arvojen tulee vastata mahdollisimman tarkasti oikeassa ympäristössä ja oikealla datalla tehtyjä testejä [Laird ja Brennan 2006]. Jotta mittauksien perusteella tuotettava data vastaisi mahdollisimman tarkasti tapaustutkimuksen kohteena olevan web CMS -järjestelmän suoritustilanteita ja olisi siten vertailukelpoista suhteessa järjestelmälle tyypillisiin suoritusarvioihin, oli testiympäristön, -datan ja -tapausten

vastattava myös järjestelmälle tyypillisiä tilanteita. Järjestelmässä ei ollut ennalta määriteltyä vakioitua testiympäristöä tai testitapauksia.

Järjestelmässä ennalta ollut testidata oli lisäksi myös jossain määrin puutteellista, ja sitä oli vähän normaaleja käyttötilanteita vastaavien testien tekemiseksi. Näin ollen oli määriteltävä testiympäristö, kasvatettava testidataa sekä määriteltävä testitapaukset. Tällöin testausta voitaisiin suorittaa myös jatkossa pitkäjänteisesti osana web CMS -järjestelmän tuotekehitystä ilman, että epämääräiset muutokset muun muassa ympäristöön, järjestelmän konfiguraatioon ja tuotteisiin tekisivät testauksesta epätarkkaa ja vertailukelvotonta eri järjestelmän versioiden välillä [Sommerville 2007].

Suorituskyvyn mittauksessa käytetyksi ympäristöksi valikoitui tapaustutkimuksen kohteena olevan web CMS -järjestelmän tyypillistä asennusympäristöä vastaava vakioitu webhotelliympäristö. Webhotelli on rakennettu CloudLinux 6 Hybrid -käyttöjärjestelmän päälle. Webhotelli hyödyntää myös LiteSpeed SAPI:a, jonka avulla PHP:ssa käytetään OPcachea. Webhotellissa on käytössä kolme kappaletta virtuaalisia suorittimia sekä enintään 4096 Mt keskusmuistia, josta 2048 Mt on varattu oletuksena PHP:n käyttöön.

Web CMS -järjestelmä asennettiin testiympäristöön yhdessä siihen liitetyn verkkokaupan kanssa. Sekä verkkokauppa- että web CMS -järjestelmästä käytettiin versiota 19.0. Testikonfiguraatio muotoiltiin vastaamaan järjestelmälle tyypillistä ja tavanomaista asennusta sekä dokumentoitiin, jotta testiympäristöä voitaisiin käyttää myös muissa yhteyksissä.

Web CMS -järjestelmän testidata sisälsi ennen täydennystä vain joitakin sisältösivuja, tuoteryhmiä, tuotteita sekä erilaisia useista tuotteista koostuvia variaatioita. Tämä tilanne ei vastannut juurikaan sellaista sisällön määrää, jota järjestelmän testaukseen voisi käyttää. Mittariston tuottamat tulokset eivät tällöin olisi olleet tarkkoja, eivätkä ne olisi antaneet todellista kuvaa järjestelmän tilanteesta [Laird ja Brennan 2006]. Järjestelmän sisältöihin lisättiin mukaan useita uusia sisältösivuja, jotta myös navigaatiohierarkiasta pystyttiin saamaan monimutkaisempi. Toiminnallisuudeltaan sisällöt ovat lähinnä tavallisia sisältösivuja, sillä suorituskyvyn ongelmien ei ole koettu sijaitsevan varsinaisesti nimenomaan sisällöistä koottavilla sivuilla, vaan verkkokauppajärjestelmän puolella.

Web CMS -järjestelmän verkkokaupan puolelle lisättiin huomattava määrä uutta testidataa. Monipuolisempi verkkokaupan testidata koostettiin lisäämällä järjestelmään ensin monitasoisempia tuoteryhmiä, minkä jälkeen tehtiin joitakin hieman raskaampia ja enemmän toiminnallisuutta sisältäviä tuotteita sekä erikokoisia useista tuotteista koostuvia variaatiotuotteita. Näiden lisäksi järjestelmään lisättiin vielä noin tuhat automaattisesti generoitavaa yksinkertaista tuotetta tasaisesti eri tuoteryhmiin, jotta järjestelmään pystyttäisiin lisäämään enemmän kuormitusta ja tarvittavan laskennan

määrää nostamaan. Testidata pystytään lisäämään järjestelmään sen hallintapaneelin kautta sekä myös poistamaan samasta paikasta.

Taulukossa 1 on kuvattu tapaustutkimuksen kohteena olevan web CMS -järjestelmän suorituskyvyn testauksessa käytetyt erilaiset kohteet. Sivut painottuvat pääasiassa verkkokauppajärjestelmän puolelle ja ainoastaan sisällönhallintaan liittyviä sivuja on vain yksi: yhdistetyissä tulosjoukoissa ”Sisältösivu 1”. Järjestelmän kaikki erilaiset tulosjoukot ja sivut sisältävät viitauksia verkkokaupasta sisältösivuihin ja toisinpäin. Täten tulosjoukot antavat myös hyvän yleiskuvan web CMS -järjestelmän erilaisten yleisten toimintojen tilanteesta.

Taulukossa 1 kuvatuissa eri tulosjoukoissa on pyritty ottamaan sellaisia yksittäisiä sivuja mukaan testitapauksiin, joissa tehdään keskenään erilaista laskentaa. Tämän seurauksena testitapaukset ovat monipuolisempia ja mittaristojen tarkkuuden tulisi säilyä paremmin [Laird ja Brennan 2006]. Esimerkiksi ”Paita”-variaatiotuotesivu on vain noin kymmenen erilaista tuotetta sisältävä variaatiotuote. Samalla taas ”Televisio”-variaatiotuotesivu on noin 60 erilaisesta tuotteesta koostuva variaatiotuote ollen samalla huomattavasti enemmän laskentaa ja käsittelyä vaativa sivu.

Sivun nimitys	Sivun tyyppi	Sivu yhdistetyissä tulosjoukoissa	Monimutkaisuus ja laskentamäärä
Alatuoteryhmä 101	Alatuoteryhmäsivu	Alatuoteryhmäsivujen keskiarvo	Yksinkertainen
Alatuoteryhmä 101 -lista	Alatuoteryhmän tuotelista	Alatuoteryhmäsivujen tuotelistan keskiarvo	Monimutkainen
Alatuoteryhmä 30301	Alatuoteryhmäsivu	Alatuoteryhmäsivujen keskiarvo	Yksinkertainen
Alatuoteryhmä 30301 -lista	Alatuoteryhmän tuotelista	Alatuoteryhmäsivujen tuotelistan keskiarvo	Monimutkainen
Frontpage	Etusivu	Etusivu	Monimutkainen
Leikkipallo	Tuotesivu	Tuotesivujen keskiarvo	Monimutkainen
Lisätuote_100_3	Tuotesivu	Tuotesivujen keskiarvo	Monimutkainen
Paita	Variaatiotuotesivu	Variaatiotuotesivujen keskiarvo	Monimutkainen
Televisio	Variaatiotuotesivu	Variaatiotuotesivujen keskiarvo	Monimutkainen
Testisivu 1	Sisältösivu	Sisältösivu 1	Yksinkertainen
Tuotteet	Päätuoteryhmäsivu	Päätuoteryhmien sivu	Yksinkertainen

Taulukko 1. Suorituskyvyn testauksessa käytetyt Web CMS -järjestelmän sivut ja sisältöjoukot.

Variaatiotuotesivujen tapaan taulukossa 1 on kuvattu myös kaksi eritasoista tavallisten tuotteiden tuotesivua: automaattisesti generoitu ”Lisätuote_100_3”-tuotesivu sekä ”Leikkipallo”-tuotesivu. ”Leikkipallo”-tuotesivu sisältää hieman enemmän lisätietoja ja hintatietoja suhteessa ”Lisätuote_100_3”-tuotesivuun. Muissa tulosjoukoissa ei ole yhtä olennaisia eroja. Sivut on jaoteltu taulukossa niiden yleisen rakenteellisen monimutkaisuuden ja tarvittavan laskennan määrän perusteella yksinkertaisiksi sekä monimutkaisiksi sivuiksi.

Kutakin testitapausta kutsutaan selaimen kautta kirjautuneena käyttäjänä siten, että ensimmäinen sivun lataus on jo suoritettu. Tällöin mahdollinen tietokannan välimuisti on jo pääosin käytössä, mikä antaa realistisemman kuvan web CMS -järjestelmän toiminnasta yhteistyössä sen kanssa. Yleismittariston avulla suoritettavien testien yhteydessä suoritetaan vähintään kymmenen kutsua jokaisen sivun kohdalla, kun taas porautuvalla mittaristolla suoritetaan vain yksi kutsu yhdellä testauskerralla. Yleismittaristolla ei kuitenkaan pystytä mittaamaan tulosjoukon ”Alatuoteryhmän tuotelista” -tuloksia kyseisen sivun AJAX-pohjaisen rakenteen vuoksi. Luvussa 5 on analysoitu erilaisia järjestelmän suorituskyvyssä testien ja mittareiden perusteella vastaan tulleita ongelmia.

5 Web CMS -järjestelmän suorituskyvyn lähtötaso ja ongelmat

Tapaustutkimuksen kohteena olevan web CMS -järjestelmän suorituskyvyn lähtötaso ei ollut kovin imarteleva mittaristoilla suoritettujen testien tuottamien tulosten perusteella. Järjestelmä kykeni suoriutumaan siihen kohdistetuista kutsuista paikoitellen hyvin pitkässä suoritusajassa, mikä ei ole edes kohtuullisella tasolla suhteutettuna Lairdin ja Brennanin [2006] yli kymmenen vuotta sitten kuvaamaan suoritusajan vähimmäistavoitetasoon, jota käyttäjät järjestelmiltä odottavat. Nykyään käyttäjien vaatima suoritustaso on luonnollisesti paljon korkeampi, joten järjestelmän saavuttamat tulokset ovat sen näkökulmasta erittäin huonoja.

Taulukossa 2 on kuvattu web CMS -järjestelmän suorituskyvyn lähtötilanne yleismittariston avulla tehtyjen testien perusteella. Tuloksista selviää, että järjestelmän suoritusajat vastaavat jossain määrin Drupalille ja Joomla!:lle tyypillisiä suoritusarvoja varsinkin monimutkaisempien sivujen osalta silloin, kun Drupalin ja Joomla!:n suorituksessa ei ole käytetty välimuistia suoritusarvojen nopeuttamiseksi [Ogunrinde ja Yoosuf 2016].

Sivu	Suoritusaika s	SQL-kyselyjen lkm	RAM-allokaatiokt
Alatuoteryhmä 101	0,205430007	1101	6144
Alatuoteryhmä 30301	0,298458767	1110	6144
Frontpage	3,358900571	2942,4	8396,8
Leikkipallo	2,574868417	2073,3	8192
Lisätuote_100_3	2,58102138	2074	8192
Paita	3,082623553	2750	8192
Televisio	4,494285297	3723,1	10240
Testisivu 1	0,181891528	87,1	6144
Tuotteet	0,184144519	99,2	6144

Taulukko 2. Web CMS -järjestelmän suorituskyvyn lähtötilanne yleismittariston perusteella.

Taulukosta 2 selviää, että yksinkertaisten sivujen kutsuminen järjestelmässä ei vie kovin paljon suoritusaikaa, vaikka niiden aikana tehtäisiinkin huomattava määrä esimerkiksi SQL-kyselyjä. Näiden sivujen osalta suorituskyky on jo nykyisellään hyväksyttävällä ja käyttäjäystävällisellä tasolla, vaikka SQL-kyselyjen lukumäärä onkin ”Alatuoteryhmä 101” ja ”Alatuoteryhmä 30301” -sivuilla varsin suuri ja siten mahdollinen merkki piilevistä ongelmista [Arsenault 2017].

SQL-kyselyjen lukumäärä on erittäin suuri jokaisen monimutkaisten sivujen joukkoon kuuluvan sivun kohdalla. Taulukosta 2 selviää, että myös järjestelmän suoritusaika on monimutkaisten sivujen kohdalla hyvin suuri: useita sekunteja jokaisen

sivun kohdalla ja pahimmillaan ”Televisio”-sivulla lähes 4,5 sekuntia. RAM-allokaation määrä ei ole kuitenkaan merkittävästi suurempi, vaikka sivujen suoritusajat ja SQL-kyselyjen lukumäärät ovatkin huomattavasti suurempia suhteessa yksinkertaisten sivujen saavuttamiin tuloksiin.

Tulosjoukko	Profilointitiedostojen keskikoko kt
Alatuoteryhmäsivujen keskiarvo	3940,5
Alatuoteryhmäsivujen tuotelistan keskiarvo	26559,5
Etusivu	36194
Tuotesivujen keskiarvo	31229,5
Variaatiotuotesivujen keskiarvo	44350
Sisältösivu 1	3520
Päätuoteryhmien sivu	3671

Taulukko 3. Web CMS -järjestelmän profilointitiedostojen koon lähtötilanne porautuvan mittariston perusteella.

Taulukossa 3 on kuvattu web CMS -järjestelmän porautuvan mittariston tuottamien profilointitiedostojen keskikoko tulosjoukoittain. Koska porautuvan mittariston profiloija kerää profilointitiedostoihin kumulatiivisesti kaikki järjestelmään kohdistuneen kutsun yhteydessä suoritetut funktiokutsut, objektien luonnit ja muut vastaavat suoritukset [Xdebug 2019], voi sen avulla arvioida näiden erilaisten tulosjoukkojen monimutkaisuutta ja vaadittavan käsittelyn määrää yleisellä tasolla.

Taulukon 3 profilointitiedostojen keskikoko noudattaa hyvin pitkälti samankaltaista trendiä taulukossa 2 kuvattujen suoritusaikojen ja SQL-kyselyjen lukumäärän osalta. Mitä monimutkaisempaa ja enemmän käsittelyä jokin sivu vaatii, sitä enemmän suoritusaikaa ja SQL-kyselyjä sen tuottamiseksi on tehty. Samalla tapaa käyttäytyvät myös Joomla! ja Drupal: kun järjestelmään kohdistuva kutsu on monimutkaisempi, vaatii se enemmän käsittelyä järjestelmältä, mikä johtaa järjestelmän suoritusajan kasvuun kyseisen kutsun suorituksessa [Ogunrinde ja Yoosuf 2016]. Taulukon 3 tuloksista erottuvat erityisesti monimutkaiset tulosjoukot, joiden profilointitiedostojen koko on huomattavan suuri suhteessa yksinkertaisiin sivuihin.

Tämän luvun alaluvuissa on eritelty tarkemmin porautuvan mittariston antamien tietojen perusteella minkälaiset tekniset ratkaisut, piirteet ja ongelmat web CMS -järjestelmän PHP-puolella aiheuttavat huonot suoritusarvot. Ongelmakohteet on havaittu myöhemmin tutkielman luvussa 6 esiteltävän iteratiivisen prosessin avulla. Oma roolini on keskeinen tässä tutkielmassa ja sen yhteydessä tehdyissä kehitystöissä: mikäli erikseen ei ole mainittu, on tämän tutkielman yhteydessä suoritetut käytännön työt ja analyysit tehty toimestani. Asiakasyrityksen muiden työntekijöiden rooli tutkimuksessa on lähinnä

erilaisten esitteltyjen ratkaisukeinojen toteutuskelpoisuuden arvioinnissa sekä tuotettujen ratkaisujen hyväksymisessä osaksi web CMS -järjestelmän yleistä versiota.

5.1 Tietokannan käsittely

Taulukosta 2 on nähtävillä, että tapaustutkimuksen web CMS -järjestelmän tietokannan käytössä on hyvin paljon vaihtelevuutta erilaisten sivujen välillä. Suhteutettuna taulukkoon 3 nähdään, että SQL-kyselyjen lukumäärä kasvaa tasaisesti sitä mukaa, kuinka raskaasta ja monimutkaisesta kutsusta on kyse. PHP-pohjaisille järjestelmille on hyvin tyypillistä, että tietokannan käyttö viekin valtaosan suoritusajasta [Arsenault 2017].

On vaikea arvioida suoraan, miksi SQL-kyselyjen lukumäärä on muotoutunut niin suureksi joillain sivuilla. On kuitenkin mahdollista, että järjestelmän parissa toimivat kehittäjät eivät ole vain kiinnittäneet asiaan huomiota, sillä tietokannan käsittelyä ei ole koettu yleisesti ongelmalliseksi järjestelmässä. PHP ei myöskään rankaise huonoista ohjelmointitekniikoista kovinkaan voimakkaasti [Cholakov 2008], jolloin tietokannan paikoitellen huono käyttötapa on voinut säilyä huomaamatta varsin pitkään. Tällöin ratkaisuja toteuttaneet kehittäjät eivät ole välttämättä edes huomanneet rakentavansa järjestelmään samalla teknistä velkaa, jonka syntymistä vahingossa esimerkiksi Buschmann [2011] on eritellyt. SQL-kyselyjen lukumäärään onkin kiinnitetty järjestelmän historiassa huomiota vain joitakin kertoja, jolloin tietokantapalvelimen ja varsinaisen web CMS -järjestelmän asennusympäristön, eli tyypillisesti webhotellin, välille on muodostunut kohtalaisen suuria vasteaikoja, jotka ovat näkyneet heti suoritusajan merkittävänä kasvamisena.

Taulukon 2 perusteella nähdään, että SQL-kyselyjä tehdään erityisen paljon juuri verkkokauppajärjestelmän osioita sisältävien kutsujen yhteydessä. Tutkimalla porautuvan mittariston avulla tuotettuja näiden sivujen profilointitiedostoja selviää, että SQL-kyselyjä tehdään hyvin paljon ympäri web CMS -järjestelmää ja pääosin verkkokauppajärjestelmän puolella. Profilointitiedostoissa ongelmallisiksi nousseita kohteita on eritelty seuraavissa kappaleissa.

Tapaustutkimuksen kohteena olevassa web CMS -järjestelmässä tehdään paljon sellaista käsittelyä, jossa on alun perin tarve luoda ja alustaa jokin tietty objektien joukko käsiteltäväksi jollakin tapaa. Tämä objektien joukko voi koostua esimerkiksi tuotteista, tuoteryhmistä tai navigaatiohierarkian soluista. Jokaista joukon solua varten tehdään tyypillisesti yksi tai useampia tietokantakutsuja, jolloin ison joukon kohdalla tietokantakutsujen määrä nousee varsin suureksi. Näitä joukkoja voidaan myös tehdä jossain muodossa useampia yhden järjestelmän kutsun yhteydessä, jolloin jo aiemmin tehtyjä tietokantakutsuja suoritetaan yhä uudestaan ja uudestaan.

Web CMS -järjestelmässä on jonkin verran järjestelmän kehyksen latauksen yhteydessä rakennettavia välimuistissa pidettäviä usein käytettyjä erilaisia tietoja, kuten kieliversioinnin ja konfiguraation tietoja. Tämä on varsin tyypillistä MVC-mallia

noudattaville järjestelmille [Wang 2011]. Tapaustutkimuksen kohteena olevassa web CMS -järjestelmässä ei kuitenkaan aina käytetä hyväksi näitä kehyksen käynnistyksen yhteydessä muotoiltavia ja haettavia tietoja, vaan osa tiedoista muodostetaan tietokannan kautta uudestaan kesken suorituksen juuri sillä hetkellä, kun tietoa tarvitaan. Myöhemmin suorituksen aikana näitä samoja tietoja voidaan jopa hakea uudestaan.

Tietokannan käyttöön liittyvät suorituskyvyn ongelmat juontavat juurensa siis kahteen erilaiseen teemaan suoritettujen mittausten perusteella: järjestelmän kehyksen kautta saatavia tietoja ei hyödynnetä tarpeeksi ja tietokannan käyttöä leimaa käyttötapa, jossa tieto haetaan juuri silloin, kun sitä tarvitaan ja unohdetaan heti sen käytön jälkeen – dataa käytetään siis tuhlailevasti.

5.2 Kehykselle keskeiset funktiot

Web CMS -järjestelmän kehykselle keskeisten funktioiden ongelmat koskevat oikeastaan kaikkia testatuista eri sivuista. Porautuvan mittariston tuottamien tuloksien avulla selviää, että järjestelmän kehyksen keskeisiä funktioita saatetaan kutsua jopa kymmeniä tuhansia kertoja yhden kutsun suorituksen aikana esimerkiksi variaatiotuotesivun ”Televisio” kohdalla. Tällöin on erittäin tärkeää, että näissä keskeisissä funktioissa käytetään vain ja ainoastaan ohjelmakoodin suorituksen näkökulmasta kaikista nopeimpia suoritustapoja, joita on listattu useissa erilaisissa PHP:n optimointia käsittelevissä julkaisuissa [Bradley 2018, Arsenault 2017, Mohammad 2017].

Keskeisten funktioiden rakenne ei ole kaikissa tapauksissa kovin suoritusystävällinen, ja ne ovat sisältäneet rakenteiltaan esimerkiksi hitaampia operaatioita, vaikka nopeammat operaatiot olisivatkin ajaneet saman asian. Tässäkin korostuu PHP:lle tyypillinen luonne: kieli ei rankaise kovin ankarasti huonojen käytäntöjen suosimisesta [Cholakov 2008]. Tällöin suorituskyvyltään heikompia ratkaisuja on voitu rakentaa myös tähän web CMS -järjestelmään huomaamatta niin, etteivät ne ole näkyneet ulospäin millään tavalla ennen ongelmien merkittävää kumuloitumista [Alcocer ja Bergel 2015].

Keskeisten funktioiden käyttöä leimaa kuitenkin niiden rakenteiden lisäksi myös alaluvussa 5.1 esitellyt tietokannan käyttöäkin koskevat ongelmat: keskeisiä funktioita käytetään pääosin ilman tarkkaa harkintaa käytön tarpeellisuudesta ja funktioiden kutsujen avulla saatujen tulosten tallentamista esimerkiksi myöhemmin käytettäväksi, vaikka tämä olisi suorituskyvyn näkökulmasta parempi ratkaisu [Arsenault 2017]. Tämä heikentää järjestelmän suorituskykyä helposti huomaamatta, vaikka ongelmakohteiden ratkaisu olisikin yksinkertaista sovelluskohteissa. Järjestelmään voi rakentua tällöin huomaamatta teknistä velkaa [Buschmann 2011].

Keskeisten funktioiden kolmas ongelmakokonaisuus johtuu tavasta, jolla niiden sisään saattaa olla rakennettuna erilaisia ylimääräisiä lisäkäsittelyjä, joita esimerkiksi vain osa järjestelmän asiakkaista käyttäisi. Tällöin näiden keskeisten funktioiden rakenne

muuttuu vähitellen sekavaksi sekä hankalasti seurattavaksi ja vaikutukset kertautuvat heti useimmissa suoritustilanteissa. Tämä onkin tyypillinen merkki ohjelmiston kokonaisarkkitehtuurin rappeutumisesta [Silva ja Balasubramaniam 2012].

Usein vaihtuvat vaatimusten määritelmät ja huonosti muuttuvia tarpeita tukeva ohjelmiston arkkitehtuuri voivat johtaa erilaisten viritysten rakentamiseen ympäri järjestelmää, vaikka ei edes tiedettäisi millä tavoin nämä muutokset vaikuttavat ympäröivään järjestelmään ja sen suoritukseen [Eick *et al.* 2001]. Tämän lisäksi keskeisten funktioiden suorituskyvyn laatuongelmia synnyttävät myös pikakorjaukset, refaktoroinnin puute ja muut vastaavat ongelmat [Eick *et al.* 2001]. Tämänkaltaisia ongelmia on havaittavissa porautuvan mittariston tuottaman datan perusteella useissa erilaisissa järjestelmän keskeisissä funktioissa ja rakenteissa.

5.3 Objektien käyttö

Objektien käyttö on pääasiassa selkeää ja suoraviivaista web CMS -järjestelmässä. Porautuvan mittariston tuottaman datan perusteella objekteja tehdään joidenkin PHP-luokkien tapauksessa yleensä vain tarvittaessa, ja niiden käytössä on huomioitu myös kohtalaisesti esimerkiksi erilaisten viitteiden käyttö ja objektien kloonaaminen uusien objektien rakentamisen sijaan, kun sellainen on mahdollista ja järkevää.

Objektien käytön ongelmat johtuvatkin oikeastaan muutamasta erilaisesta teemasta, joista ensimmäinen jatkaa hyvin pitkälti samalla linjalla jo aiemmin alaluvuissa 5.2 ja 5.1 eriteltyjen havaintojen kanssa. Web CMS -järjestelmässä käytetään osaa luokista luotavia objekteja varsin tuhlailevasti. Esimerkiksi MVC-mallin mukaisia tietokannan malli-luokkia käytetään hyvin usein ilman luokkien kierrättämistä seuraavaan käyttökohteeseen tai jopa luomalla uusi identtinen objekti jokaisella jonkin luupin kierroksella. Suorituskyvyn kannalta tämä ei ole järkevää, sillä objektien luonti on PHP:ssä aina jonkin verran ylimääräistä laskentaa vaativa toiminto [Cholakov 2008].

Objektien paikoitellen tuhlailevan käytön lisäksi luokkien alustuksissa tehtävän laskennan määrä vaihtelee hyvin paljon eri luokkien välillä. Jonkin objektin luonti ei vaadi web CMS -järjestelmässä juuri yhtään laskentaa, mutta monimutkaisempien ja enemmän toiminnallisuutta sisältävien objektien luonti voi vaatia huomattavan määrän jo objektin luonnin yhteydessä tehtävää laskentaa. Tällaisia suuria ja laajalle ulottuvia monimutkaisia luokkia on kutsuttu muun muassa *jumalluokiksi* (God classes) [Fokaeds *et al.* 2012]. Näiden luokkien objektien tuhlailevalla käytöllä voi olla merkittäviä vaikutuksia järjestelmän suorituskyvyn kannalta.

Objektien käytössä on niiden tuhlailevan käytön lisäksi myös muita ongelmia. Osa web CMS -järjestelmän luokista on jaoteltu erilaisiin alustuksen tasoihin. Objektin luonnin jälkeen suoritetaan näiden luokkien tapauksessa yleensä objektin tiedot alustava funktio, jota objektin rakentajan yhteydessä ei vielä suoriteta. Alustusfunktion parametrisoinnista tai kutsutusta funktiosta riippuen suoritetaan yleensä vaihteleva määrä

laskentaa ja erilaisten tietojen hakemista. Esimerkiksi tuote-luokka voidaan alustaa tuotesivua tai tuotelistaa varten, jolloin suoritettavan laskennan määrä muuttuu erittäin merkittävästi.

Web CMS -järjestelmän porautuvan mittariston tuottamasta datasta selviää, että esimerkiksi näitä tuote-luokkia alustetaan järjestelmän eri osissa tuotelistaa ja tuotesivua lukuun ottamatta jossain määrin miten sattuu ilman tarkkaa jaottelua sen välillä, koska käytetään suppeampaa ja koska laajempaa luokan alustusta. Tämän seurauksena esimerkiksi etusivun suoritusajasta huomattava osa kuluu juuri tuote-luokkien objektien käsittelyyn.

Vaikka osaa web CMS -järjestelmän luokista käytetäänkin järkevästi järjestelmässä, kertovat objektien käytön ongelmat järjestelmässä piilevästä teknisestä velasta sekä ohjelmiston rappeutumisesta [Eick *et al.* 2001]. Järjestelmän arkkitehtuuri on myös osaltaan sallinut objektien hyvin vaihtelevan käytön, sillä joissakin osissa suoritettavia toimintoja ei ole abstraktoitu kovin korkealle tasolle. Tämä on mahdollistanut toimintoja käyttöön ottavalta kehittäjältä virheiden tekemisen luokkien objektien alustamisessa ja käyttämisessä eri sovelluskohteissa. Tällöin järjestelmään on syntynyt helposti tahatonta teknistä velkaa [Buschmann 2011].

5.4 Tietorakenteiden ongelmat

Web CMS -järjestelmän tietorakenteiden ongelmat koskevat porautuvan mittariston avulla saadun datan perusteella vain joitakin järjestelmän osia. Tietorakenteiden osalta ongelmana korostuu paikoitellen liian monimutkaiset ja raskaat tietorakenteet, joita on hankala seurata ja huomioida järjestelmän kehityksessä. Esimerkiksi sisältösivuista luotava navigaatio sisältää useita erilaisia luokkia, lukuisia sisäkkäisiä funktioita, paljon yksinkertaisia luuppien sisällä tapahtuvia tietokantakyselyjä sekä muun muassa erilaisia näkyvyyksien tarkistuksia useiden erilaisten funktioiden kautta. Tällöin näiden järjestelmän osien muokkaaminen voi olla hankalaa, sillä kehittäjän tulee tuntea kaikki muutkin osiot, joihin muokattavat osat vaikuttavat, jotta järjestelmän rakenne ei samalla rappeutuisi [Eick *et al.* 2001].

Web CMS -järjestelmän tietorakenteiden ongelmat koskevat pääasiassa järjestelmän vanhempia osia, joita edellä kuvattu navigaation rakennuksen esimerkkikin edustaa. Uudemmissa järjestelmän osissa tietorakenteet on muotoiltu abstraktimmiksi, yksinkertaisemmiksi ja vähemmän erilaisia hankalasti seurattavia ja monimutkaisia suorituspolkuja sisältäviksi kokonaisuuksiksi, jolloin niiden laajentaminen ja ylläpitäminen on helpompaa [Eick *et al.* 2001]. Tämä näkyy myös porautuvan mittariston tuloksissa, joissa esimerkiksi pääosin uudempaa koodipohjaa edustavan web CMS -järjestelmän verkkokauppajärjestelmän sisältämät erilaiset tietorakenteet ovat huomattavasti yksinkertaisempia suhteessa ongelmallisiin järjestelmän osiin.

Web CMS -järjestelmä hyödyntää sivujen tulostamisessa sivupohjamoottoria, jossa lopullinen tulostettava sivu muotoillaan dynaamisesti kutsun yhteydessä staattisten, mutta PHP:n avulla parametrisoitavien ja sisällöltään vaihtelevien, HTML-sivujen pohjalta. Tämän sivupohjamoottorin rakenne noudattaa jossain määrin Komara *et al.* [2016] ja Cu *et al.* [2009] kuvaamia sivupohjamoottoreita, joissa sivupohjamoottorin avulla pystytään erottamaan PHP-koodi sekä sivupohjat täysin omiksi ja eriytetyiksi tiedostokokonaisuuksikseen.

Web CMS -järjestelmän sivupohjamoottori edustaa pääosin järjestelmän vanhempia tietorakenteita, vaikka sitä onkin paranneltu ja laajennettu vähitellen järjestelmän elinkareen aikana. Porautuvan mittariston perusteella selviää, että sivupohjamoottorin kieliversiointi, sen luomat lukuisat sivupohja-objektit sekä sivupohjien osioiden objektit ja sivupohjien yleisten parametrien asetus sisältävät jonkin verran ongelmia suorituskyvyn näkökulmasta. Näin ollen web CMS -järjestelmän sivupohjamoottorin useat eri toiminnot vievät merkittävästi suoritusaikaa. Sivupohjien yleisiä parametreja on myös asetettu useissa eri järjestelmän kohteissa, mikä altistaa niiden asetuksen turhalle toistolle kutsujen yhteydessä.

5.5 Kokonaisarkkitehtuurin rappeutuminen

Web CMS -järjestelmän suorituskyvyn ongelmat ovat varsin monimuotoisia. Tarkastelemalla järjestelmän ohjelmakoodia sekä yleismittariston tuottamia järjestelmän yleisiä suoritusarvoja ja porautuvan mittariston seikkaperäistä tietoa voidaan kuitenkin todeta järjestelmän suorituskyvyn perusongelman johtuvan hyvin perustavanlaatuisesta ongelmasta – järjestelmän kokonaisarkkitehtuurin rappeutumisesta.

Ohjelmiston kokonaisarkkitehtuurin rappeutumista on eritelty tarkemmin ilmiönä tämän tutkielman luvussa 3. Web CMS -järjestelmässä kokonaisarkkitehtuurin rappeutuminen ilmenee kuitenkin usein erilaisin tavoin. Järjestelmän sisällönhallintaosion ja verkkokauppajärjestelmän ohjelmakoodi sekä erinäiset funktiot ja luokat ovat osittain sekoittuneet, vaikka verkkokauppajärjestelmän tulisi olla lähtökohtaisesti vain sisällönhallintajärjestelmän lisämoduuli. Tämä tekee järjestelmän ylläpidosta hankalampaa ja toisaalta lisää tarvittavan osaamisen ja ajan määrää, jotta järjestelmää voi muokata, ylläpitää ja jatkokehittää onnistuneesti [Rocha *et al.* 2017].

Kokonaisarkkitehtuurin rappeutuminen näkyy web CMS -järjestelmässä myös siinä, ettei sen ohjelmakoodista ole poistettu järjestelmällisesti jo poistuneiden yrityksen asiakkaiden räätälöintejä. Tällöin kaikkien yksittäisten hyvinkin kapseloitujen toimintojen vaikutukset näkyvät kuitenkin myös koko järjestelmän suoritusarvoissa sekä tekevät ylipäätään ohjelmakoodin rakenteesta hankalammin ylläpidettävän. Järjestelmille onkin tyypillistä, että ne muuttuvat ajan myötä vähitellen monimutkaisemmiksi, vaikeammin ylläpidettäviksi sekä hankalammin ymmärrettäviksi [Neill ja Laplante 2006]. Porautuvan mittariston tulosten perusteella selviää, että esimerkiksi yleistä

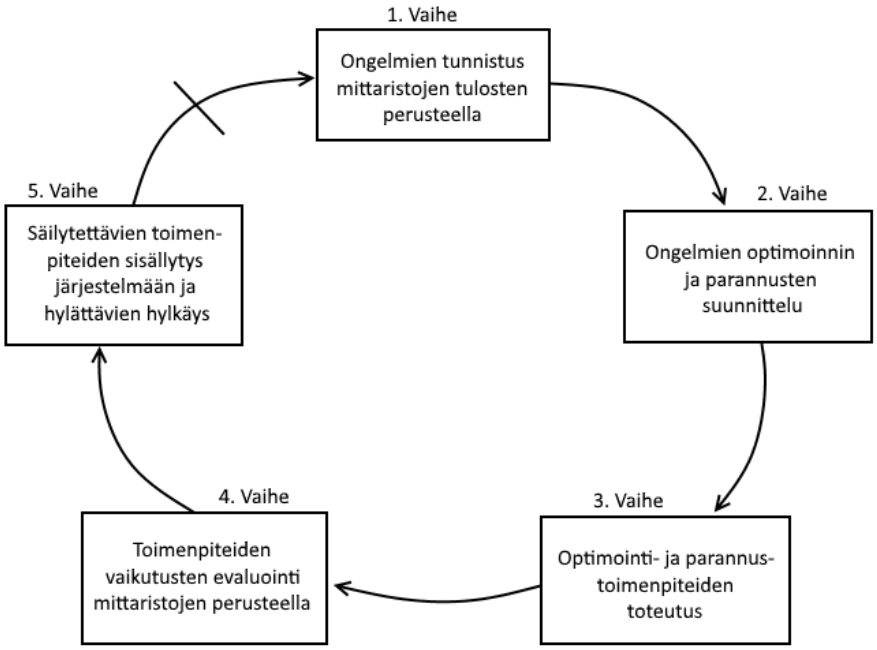
asetusten funktiota suoritetaan järjestelmässä useita tuhansia kertoja jokaisen kutsun yhteydessä. Näistä suorituskerroista huomattava osa syntyy juuri asiakaskohtaisten toimintojen suorituksien tarkastelusta.

Web CMS -järjestelmän kokonaisarkkitehtuurin rappeutumista korostavat myös alaluvuissa 5.1, 5.2, 5.3 sekä 5.4 tehdyt huomiot järjestelmän suorituskykyä heikentävistä ongelmista. Nämä erinäiset seikat voivat johtua siitä, että järjestelmää on kehitetty eteenpäin voimakkaalla tahdolla ilman, että samalla olisi ylläpidetty ja toisaalta myös tarkkailtu kokonaisarkkitehtuurin kehitystä koko järjestelmän laadun näkökulmasta. Pitkän aikavälin myötä kumuloituneet laiminlyönnit kokonaisarkkitehtuurin kehityksessä ja huomioinnissa voivat olla hankalia poistaa, kun ne lopulta huomataan suorituskyvyn laskun myötä [Alcocer ja Bergel 2015]. Onkin siis keskeistä, että suorituskykyä parannettaessa otetaan myös huomioon järjestelmän kokonaisarkkitehtuurin kehitys ja sen tarvitsema huomio, eikä keskitytä esimerkiksi lyhyen tähtäimen hyötyjä tuottaviin yksinkertaisiin ratkaisuihin. Tutkielman luvussa 6 käsitellään erilaisia ratkaisuja, joiden myötä järjestelmän suorituskykyä on pyritty parantamaan tämän tutkielman yhteydessä yleisesti hyväksi havaittujen keinojen kautta. Luvussa kuvataan myös prosessi, jota erilaisten järjestelmän suorituskyvyn ongelmien löytämiseksi sekä parannus- ja optimointiratkaisujen valitsemiseksi, tuottamiseksi ja evaluoimiseksi on käytetty.

6 Web CMS -järjestelmän suorituskyvyn parantaminen ja optimointi

Tapaustutkimuksen kohteena olevan web CMS -järjestelmän suorituskyvyn parantamisessa ja optimoinnissa pyrittiin löytämään sellaisia ratkaisuja, joiden avulla järjestelmän kokonaisarkkitehtuurin tilannetta pystyttäisiin myös parantamaan. Sopivien ratkaisujen löytämiseksi hyödynnettiin ja sovellettiin aktiivisesti erilaisia ratkaisuja, jotka on jo aiemmin havaittu toimiviksi jossain muussa yhteydessä. Ratkaisuissa huomioitiin erilaiset PHP:n parhaimmiksi havaitut käytännöt kuin myös teoreettisemmat vaihtoehdot sekä järjestelmän muu arkkitehtuuri ja erilaiset siihen kohdistuvat toimintaympäristön vaatimukset.

Jotta kunkin yksittäisen suorituskyvyn parantamiseksi ja optimoimiseksi tehdyn kehitystoimenpiteen vaikutuksia voitaisiin arvioida web CMS -järjestelmässä, suoritettiin kehitystoimenpiteet iteraatioissa. Tämä mahdollisti myös yleismittariston ja porautuvan mittariston tulosten seuraamisen jokaisen iteraation yhteydessä. Iteratiivisessa kehityksessä onkin tarkoitus pystyä erottelemaan tavoitteiden kannalta huonoiksi osoittautuvat kehitystoimenpiteet hyvistä ratkaisuista [Sotirovski 2001]. Tällöin on tärkeää, että voidaan nopeasti hylätä ongelmalliseksi koetut kehitystoimenpiteet ja säilyttää hyviksi havaitut seuraavia iteraatioita varten, jolloin niitä voidaan myös tarvittaessa edelleen jatkokehittää [Sotirovski 2001]. Lopulta iteratiivisen prosessin myötä on todennäköistä löytää toimiva ratkaisu esitettyyn ongelmaan [Hevner *et al.* 2004].



Kuva 1. Web CMS -järjestelmän suorituskyvyn iteratiivinen parantaminen ja optimointi.

Kuvassa 1 on kuvattu web CMS -järjestelmän suorituskyvyn parantamisessa ja optimoinnissa tämän tutkielman yhteydessä käytetty iteratiivinen kehitysprosessi. Vaikka vaiheita on kuvattu kuvassa 1 selkeiden vaihdoksien avulla, ovat eri vaiheet todellisuudessa osittain toisiinsa limittyviä. Iteratiivisessa kehitysmallissa ei ole yleensä tarkkaa rajaa erilaisten vaiheiden välissä [Sommerville 2007]. Kuvassa 1 vaiheiden rajoja on käytetty lähinnä kehitysprosessin luonnollisen kulun havainnollistamiseksi selkeämmin.

Kehitysprosessin ensimmäisessä vaiheessa suoritetaan suorituskyvyn testit web CSM -järjestelmässä sekä yleisellä että porautuvalla mittaristolla. Mittaristojen tulosten perusteella pyritään löytämään kyseisen iteraation kohdalla järjestelmässä eniten esiin nousevia ongelmallisia kohteita. Kyseisessä iteraatiossa pyritään kehittämään ratkaisuja juuri näitä havaittuja ongelmallisia kohteita varten. Erityisesti porautuvan mittariston rooli on keskeinen kunkin iteraation ensimmäisessä vaiheessa, sillä sen avulla pystytään tarkkailemaan suorituskyvyn ongelmia myös yksittäisten funktioiden tasolla yleisten arvojen sijaan [Xdebug 2019].

Toisessa vaiheessa suunnitellaan keinot ja tavat, joilla havaittuja ja parannettavaksi sekä optimoitavaksi valittuja ongelmakohteita parannetaan. Tässä vaiheessa on hyödynnetty keskeisesti erilaisia muissa yhteyksissä käytettyjä ja löydettyjä keinoja, joilla suorituskykyä voidaan parantaa huomioiden samalla järjestelmän kokonaisarkkitehtuurin kehitys. Tässä vaiheessa on otettu myös huomioon mahdollisuus kokeilla erilaisia vaihtoehtoja prototyyppien muodossa. Prototyypit mahdollistavatkin tutkailevan iteratiivisen prosessin parhaiden keinojen löytämiseksi [Goldman ja Narayanaswamy 1992].

Toisen vaiheen aikana suunnitellut kehitystoimenpiteet toteutetaan vaiheessa kolme, jossa keskeisenä osana on ottaa huomioon erilaisten ratkaisujen soveltuvuus järjestelmään ja rakennettavien ratkaisujen vaikutukset järjestelmän toiminnallisuuteen. Teknistä velkaa voidaan käsitellä joko maksamalla se pois, muokkaamalla se helpommin hallittavaan muotoon tai maksamalla jatkuvasti korkoa siitä aiheutuvista ongelmista [Buschmann 2011]. Iteratiivisen prosessin kolmannessa vaiheessa pyritäänkin myös vaikuttamaan järjestelmässä olevaan tekniseen velkaan vähentämällä sitä tai muokkaamalla sitä helpommin hallittavissa olevaan muotoon toteutettavien kehitystoimenpiteiden seurauksena.

Neljännessä vaiheessa evaluoidaan toteutetut kehitystoimenpiteet web CMS -järjestelmään rakennettujen suorituskyvyn mittaristojen avulla. Mikäli jokin järjestelmään toteutettu ratkaisu ei tuota tuloksissa havaittavaa parannusta kaikilla tai joillakin mittaristoilla mitatuilla arvoilla, on toteutettu ratkaisu hylättävä, vaikka vaihtelu onkin tyypillistä suorituskyvyn kehitykselle [Alcocer ja Bergel 2015]. Rakennettujen

ratkaisujen tulisi tuottaa mittaristoilla havaittavia parannuksia ja mahdollisuuksien mukaan myös vähentää järjestelmän kompleksisuutta ja parantaa sen ylläpidettävyyttä.

Viidennessä iteraation vaiheessa onnistuneet kehitystoimenpiteet sisällytetään hallitusti web CMS -järjestelmään. Jotta järjestelmään sisällytettäviä kehitystoimenpiteitä ei liitettäisi vain sokeasti järjestelmään, validoidaan ne vielä erikseen järjestelmän parissa työskentelevien muiden kehittäjien toimesta. Tämän avulla pystytään varmistamaan, ettei kehitystoimenpiteissä sokeuduta keskittymään vain mittareiden avulla saatujen lukujen parantamiseen [Laird ja Brennan 2006] ja ettei kehitystoimenpiteiden myötä luoda uutta teknistä velkaa web CMS -järjestelmään [Rocha *et al.* 2017]. Viidennen vaiheen päättyessä aloitetaan uusi iteraatio taas vaiheen yksi kautta, jolloin keskitytään etsimään edellisen iteraation parannusten jälkeen uusina ongelmakohteina esiin nousevia asioita.

Kaikkiaan web CMS -järjestelmän suorituskyvyn parantamiseksi ja optimoinniksi tehtiin yhteensä 12 kehitysiteraatiota, joiden aikana keskityttiin hyvin erilaisiin järjestelmän osioihin ja ongelmakohteisiin. Iteraatioiden aikana käsiteltiin kaikkia tutkielman luvussa 5 kuvattuja erilaisia järjestelmän suorituskyvyssä ongelmalliseksi havaittuja eri kohteita ja pyrittiin kehittämään niihin ratkaisuja. Iteraatioiden määräksi muotoutui 12 lähinnä käytettävissä olevan ajan sekä saavutettujen tuloksien takia. Kehitysiteraatiot jakautuivat tasaisesti tammi- ja huhtikuun välille talven ja kevään 2019 aikana.

Tämän luvun alaluvuissa käsitellään tapaustutkimuksen kohteena olevan web CMS -järjestelmän suorituskyvyn parantamiseksi tehtyjä erilaisia ratkaisuja. Ratkaisuja käsitellään yksittäisten iteraatioiden sijaan yhteisesti löydettyjen ratkaisujen pääteemojen kautta kaikkien 12 eri iteraation osalta.

6.1 Kapselointi, käsittelyn keskittäminen ja toiston purkaminen

Tutkielman luvussa 5 on eritelty useita erilaisia sellaisia web CMS -järjestelmän suorituskyvyn ongelmia, jotka ovat johtuneet datan tuhlailevasta käytöstä, päällekkäisyyksistä, toistosta tai toisaalta esimerkiksi huonosti parametrisoiduista ja vaihtelevasti käytetyistä luokkien objekteista. Osa näistä luokista on jumalluokiksi kutsuttuja hyvin laajoiksi ja monipuoliseksi kasvaneita kokonaisuuksia [Fokaeds *et al.* 2012].

Web CMS -järjestelmän tapauksessa nämä suuret jumalluokat sisältävät erittäin suuren määrän järjestelmälle keskeisiä toimintoja, kuten yksittäisen verkkokauppajärjestelmän tuote-luokan kohdalla suoraan tai muiden tuote-luokan attribuutteina toimivien luokkien kautta lähes kaiken mahdollisen, mitä tuotteen toimintoihin kuuluu. Web CMS -järjestelmän jumalluokat ovat siis lähinnä *toiminnallisia jumalluokkia* (behavioral god classes) [Fokaeds *et al.* 2012]. Tällaisissa tapauksissa rakennetta voidaan parantaa siirtämällä luokan toimintoja lähemmäksi niiden käyttökohdetta tai vastaavasti keräämällä niistä selkeitä kokonaisuuksia ja tuottamalla

näistä uusia kapseloituja luokkia [Fokaeds *et al.* 2012]. Web CMS -järjestelmän tapauksessa ongelmat johtuvat kuitenkin ainakin osittain tavasta, jolla näitä suuria luokkia käytetään. Tällöin niiden purkaminen pienempiin osiin ei poista ainakaan suoraan mahdollisuutta käyttää kyseisiä luokkia väärin.

Jotta jumalluokkia ei voisi enää käyttää niin helposti väärin web CMS -järjestelmässä, pyrittiin niitä kutsuvia kohteita kapseloimaan Fowler *et al.* [1999] kuvaamilla tavoilla abstraktoiduiksi omiksi luokikseen monissa eri paikoissa toistuvien kymmeniä koodirivejä pitkien käsittelyjen sijaan. Esimerkiksi järjestelmän verkkokauppajärjestelmän tuotenostoissa esiintyi merkittävästi hajontaa tuote-luokkien parametrisoinnissa, minkä seurauksena niiden suoritusluvut olivat hyvin vaihtelevia porautuvan mittariston mukaan. Abstraktoituja luokkia päätettiin muotoilla polymorfian avulla, jolloin näiden yläluokkien tulisi myös toteuttaa erilaisten toistuvien toimintojen tarpeita esimerkiksi yksinkertaisen parametrisoinnin kautta [Tsantalis ja Chatzigeorgiou 2010]. Tämä mahdollistaa myös abstraktoitujen luokkien jatkokehittämisen ilman niin suurta riskiä ohjelmiston rappeutumisesta, kun ne tukevat jo lähtökohtaisesti erilaisia esiin nousevia tarpeita ja niiden huomioon ottamista [Eick *et al.* 2001].

Abstraktoitujen luokkien avulla toteutetun kapseloinnin avulla pystyttiin vaikuttamaan tehokkaasti siihen, että näitä jumalluokkia alustetaan objekteiksi oikeiden käyttötilanteeseen sopivien parametrien avulla. Tämä myös vähentää toistuvaa ylläpitotarvetta merkittävästi, kun aiempien useiden erilaisten kohteiden sijaan on tarve ylläpitää vain yhtä yksittäistä toiminnot toteuttavaa yläluokkaa, kun toimintoja laajennetaan ja muokataan [Tsantalis ja Chatzigeorgiou 2010].

Kapselointi ratkaisi kuitenkin vain osan web CMS -järjestelmän päällekkäisyydestä, toistosta ja datan tuhlailevasta käytöstä kumpuavista suorituskyvyn ongelmista. Ongelmana olikin edelleen se, että samankaltaisia toimintoja ja muun muassa yleisiä sivupohjiin asetettavia muuttujia suoritettiin useissa eri kohteissa toistuvasti yhden kutsun aikana. Pääosin nämä ongelmat johtuivat todennäköisesti järjestelmän kokonaisarkkitehtuurin noudattamatta jättämisestä ja toisaalta myös tahattomasta tietämättömyydestä siitä, että näitä toimintoja voitaisiin suorittaa esimerkiksi kootusti yhden kerran jonkin kutsun aikana – tähän ei vain ollut kiinnitetty huomiota.

Yleistä toistoa pyrittiin purkamaan järjestelmästä siirtämällä näitä toistuvia ja päällekkäisiä kutsuja yhdellä kertaa jonkin keskeisen web CMS -järjestelmän yleisen metodin yhteydessä suoritettaviksi. Esimerkiksi sivupohjiin asetettavia yleisiä muuttujia varten järjestelmässä oli jo keskitettyjä funktioita, jolloin paikka näille oli valmiiksi myös olemassa. Esimerkiksi Fowlerin ja muiden [1992] -teoksessa puhutaan duplikaattiohjelmakoodin eriyttämisestä esimerkiksi uusiksi ylätason luokiksi merkittävänä toistoa vähentävänä keinona. Samalla tapaa suorituksen aikana usein

suoritettavia toimintoja on hyvä pyrkiä yhdistämään mahdollisuuksien mukaan esimerkiksi vain kerran suoritettaviksi toiminnoiksi, mikäli se on mahdollista.

Toistoa purettiin ylemmän tason muutoksien lisäksi myös mikrotasolla muokkaamalla usein toistuvia toimintoja esimerkiksi sellaisiksi, että luupin sisällä tarvittavia ja sen aikana muuttumattomia arvoja haetaan ennen luuppeja muuttujiin tallennettaviksi, jolloin luupin sisällä pystyttiin viittaamaan vain tähän jo ennalta haettuun tietoon. Erityisesti usein toistuvien luokkakohtaisten toimintojen kohdalla tämä osoittautui tehokkaaksi keinoksi vähentää turhaan toistuvaa ja päällekkäistä laskentaa eri puolilla web CMS -järjestelmää. Muuttujien käsittely on erittäin helppoa ja yksinkertaista PHP:ssä [Cholakov 2008], joten web CMS -järjestelmässä on ollut helppo käyttää muuttujia ja toisaalta myös yksinkertaisten funktioiden tuloksia kertakäyttöisinä kohteina. Toistoa purettiin myös kokoamalla luuppien sisällä toistuvia yksittäisiä tietokantakutsuja yhdellä kertaa suoritettaviksi. Tällöin luupeissa voidaan käyttää jo ennalta haettua tietoa sen sijaan, että esimerkiksi kukin luupin sisällä alustettava objekti suorittaisi samat tietokantakyselyt yksittäisinä kutsuina. Tämä on keskeistä, sillä tietokantakyselyt voivat viedä huomattavan osan PHP-pohjaisen järjestelmän suoritusajasta [Arsenault 2017, Bradley 2018].

Suoran suorituskyvyn parannuksen lisäksi turhan toiston vähentymisen myötä myös usein käytettyjen muuttujien käyttö ja muunneltavuus helpottuu, kun ylläpidettävien kohteiden määrä vähenee ohjelmakoodissa. Teknisen velan määrän laskun myötä myös ohjelmiston laatu paranee ja kehitystöiden vaatima aika laskee [Rocha *et al.* 2017].

Web CMS -järjestelmän usein toistuvien käsittelyjen keskittämisessä oli myös tärkeää huomioida kehyksen yhteydessä suoritettavien yleisten toimintojen suorituskyky ja -määrät. Kehyksen on tarkoitus esimerkiksi suorittaa järjestelmään kohdistuneiden kutsujen tulosten osalta kieliversiointi vain kerran yhden kutsun aikana, mutta porautuvan mittariston perusteella kieliversioinnin suoritusten määrässä oli jonkin verran hajontaa suoritetusta kutsusta riippuen. Web CMS -järjestelmästä purettiin pois nämä turhat ylimääräiset kutsut kehyksen toiminnoista, jolloin järjestelmän kehyksen vaatima osuus suorituksista pienenee ja koko järjestelmän suorituskyky paranee kutsujen tehokkuuden kasvun myötä [Wang 2011].

6.2 Välimuistin käytön lisääminen ja objektien käytön tehostaminen

Tutkielman luvussa 5 on eritelty tapaustutkimuksen kohteena olevassa web CMS -järjestelmässä suoritettavien SQL-kyselyjen lukumääriä ja objektien käsittelyä. Kyselyjä tehdään yleismittariston tuottamien tietojen perusteella erittäin suuria määriä yksittäisten kutsujen kohdalla. Lisäksi järjestelmässä käytetään objekteja paikoitellen varsin tuhlailevasti. Näiden toimintatapojen seurauksena porautuvan mittariston tuottamassa

materiaalissa korostuukin web CMS -järjestelmän erilaisilla sivuilla tietokantakyselyihin sekä uusien objektien luontiin kuluva suoritusaika.

Joomla!:n ja Drupalin suoritusajat paranevat merkittävästi, kun järjestelmien kutsujen yhteydessä hyödynnetään jonkinlaista välimuistiin pohjautuvaa tekniikkaa. Erityisesti Drupalin tapauksessa suorituskyvyn nousu välimuistin ollessa käytössä on erittäin suuri. [Ogunrinde ja Yoosuf 2016] Tapaustutkimuksen kohteena olevassa web CMS -järjestelmässä tietokantakutsujen käyttöön on jo olemassa välimuistiin pohjautuva PHP:n sessiota hyödyntävä järjestelmä, mutta sitä ei käytetä juurikaan tietokantakutsujen yhteydessä. Muita tapoja käyttää tietokannan välimuistia olisivat esimerkiksi tulosten tallentaminen väliaikaisesti tiedostoihin tai erillisten välimuistimoottoreiden hyödyntäminen [Sa'adah *et al.* 2015].

Tietokannan välimuistin käyttöä päätettiin laajentaa merkittävästi tietokantakutsujen vähentämiseksi sen jälkeen, kun alaluvussa 6.1 esitellyt toiston vähentämiseksi suoritetut toimenpiteet oli tehty. Välimuistin käyttö on erittäin tehokas keino kasvattaa web-pohjaisten järjestelmien suorituskykyä [Sa'adah *et al.* 2015]. Web CMS -järjestelmän suorituskyvyn parantamisessa päätettiin käyttää järjestelmässä jo olevaa tietokannan välimuistin rakennetta.

Tietokannan välimuistin käyttöönotossa eri puolilla web CMS -järjestelmää oli oltava tarkka: kaikki tietokantakyselyt eivät sovellu välimuistin turvin käytettäväksi. Esimerkiksi sellaiset tilanteet, joissa jokaisen kutsun yhteydessä on aina saatava ajan tasalla oleva tieto, eivät sovellu välimuistin avulla käytettäväksi. Tällaisia kohteita web CMS -järjestelmässä ovat muun muassa verkkokauppapuolen tuotteiden saldomäärät. Valituissa kohteissa tietokannan välimuisti luotiin siten, että siihen kerättiin kaikki kunkin vaihtelevan kutsun yksilöintiä varten tarvittavat parametrit ja hakutulokset ensimmäisellä SQL-kyselyn suorituskerralla. Seuraavien kyselyjen yhteydessä voitiin viitata tähän jo haettuun tietoon välimuistin voimassaolon ajan.

Web CMS -järjestelmä muodostaa usein jonkin asiakasyrityksen verkkosivut ja on siten tärkeää, että sen hallinta säilyy heidän käsissään [Grubor ja Jakša 2018]. Tällöin tietokannan välimuistin käytön määrän kasvaessa sen on myös säilyttävä sellaisena, että sitä pystytään tarvittaessa hallitsemaan. Tietokannan välimuisti olikin rajattava siten, että se nollautuisi esimerkiksi käyttäjän kirjautumisen yhteydessä ja niin, että se oli manuaalisesti ohitettavissa yksittäisen kutsun yhteydessä tai nollattavissa kokonaan sekä otettavissa kokonaan pois käytöstä tarvittaessa.

PHP-tiedostot käännetään suorituksen yhteydessä alemman tason ohjelmakoodiksi. PHP:n tapauksessa jo suoritettujen ja käännettyjen tiedostojen välimuistina voidaan käyttää PHP:n laajennosta OPcachea. Se tallentaa jo aiemmin käännetyn ohjelmakoodin talteen muistiin ja seuraavan kutsun yhteydessä PHP tarkastaa onko tämän jo käännetyn ohjelmakoodin tiedostot muuttuneet suhteessa edelliseen

suoritukseen. Mikäli ne eivät ole muuttuneet, voidaan jo valmiiksi esikäännettyä ohjelmakoodia käyttää suorituksessa. [Cholakov 2008, Arsenault 2017] Web CMS -järjestelmässä OPcache on jo käytössä pääosassa sen tyypillisiä webhotelliympäristöjä, joten sen tuoma suorituskyvyn hyöty näkyy jo järjestelmän suorituskyvyn lähtötilanteessa myös tässä tutkielmassa käytetyssä testiympäristössä ja suoritetuissa testeissä.

Välimuistin käytön toiseksi keskeiseksi käyttökohteeksi valittiin web CMS -järjestelmän sivupohjamoottori, jonka tapa käyttää sivupohjia osoittautui tutkielman luvussa 5 esitellyin tavoin suorituskykyä hidastavaksi. Sivupohjamoottori käyttää web CMS -järjestelmille Boikon [2005] kuvaamalla tavalla tyypilliseen tapaan staattisia HTML-sivuja hyvin monia kertoja saman kutsun aikana ja luo näiden käyttöä varten sisältöobjekteja sekä sisältöosien objekteja. Kaikkia sivupohjia ei ole kuitenkaan tarve tallentaa välimuistiin, sillä joitakin sivupohjia käytetään vain yhden kerran yksittäisen web CMS -järjestelmään kohdistuvan kutsun yhteydessä. Näin ollen välimuisti kohdistettiin vain sellaisiin sivupohjiin, joita kutsutaan useita kertoja yhden järjestelmään kohdistuvan kutsun aikana.

Nämä toistuvat sivupohjat kartoitettiin porautuvan mittariston tuottamien tuloksien avulla tarkkailemalla sivupohjien toistuvia käyttökohteita web CMS -järjestelmän sisällä ja tarkastelemalla näiden käyttämiä sivupohjia. Sivupohjista kerättiin sellaiset sivupohjat, jotka toistuvat selkeästi usein kutsujen yhteydessä. Tällaisia sivupohjia olivat muun muassa navigaatioon ja tuotelistauksiin liittyvät sivupohjat. Nämä sivupohjat voivat muuttua kutsujen välillä, joten sisältösivujen välimuisti toteutettiin siten, että sivupohjaobjektin koko sisältö tallennettiin yhteen välimuistin soluun ensimmäisen objektin luonnin yhteydessä. Tämän tyhjän, mutta alustetun, sivupohjaobjektin tietoja haetaan sitten uusille vastaaville sivupohjaobjekteille niiden luonnin yhteydessä tietojen uudelleenkäsittelyn sijaan. Dynaamisille web-pohjaisille järjestelmille on tyypillistä, että ne ovat hitaampia kuin staattisista sivuista koostuvat järjestelmät [Boiko 2005], mutta tällä sivupohjien välimuistilla pystytään aktiivisesti laskemaan dynaamisuuden tuottamaa suorituskyvyn laskua erityisesti web CMS -järjestelmän laajempien sivukokonaisuuksien kohdalla.

Välimuistien käytön lisäämisen lisäksi myös objektien käyttöä tehostettiin muokkaamalla tapaa, jolla usein käytettyjä luokkia käytetään järjestelmässä. Objektien luonti itsessään sisältää PHP:ssä jonkin verran ylimääräistä laskentaa [Cholakov 2008], mutta tämän lisäksi myös objektien luonnin yhteydessä suoritettavat muut alustuksen toiminnot vievät web CMS -järjestelmässä merkittävästi suoritusaikaa ja aiheuttavat esimerkiksi lukuisia SQL-kyselyjä porautuvan mittariston perusteella.

Joidenkin luokkien tapauksessa ratkaisuksi haettiin näiden luokkien tallentamista globaaleiksi objekteiksi jo järjestelmän kehyksen käsittelyn yhteydessä kunkin kutsun suorituksen alussa. Globaalien muuttujien laajaa käyttöä ei pidetä PHP:n kohdalla

yleisesti kovin järkevänä [Mohammad 2017, Bradley 2018], mutta tässä tapauksessa niiden tuottama hyöty ajoi mahdollisten haittojen edelle. Globaaleihin muuttujiin tallennettiin muun muassa useita tietokannan käsittelystä vastaavia malli-luokkia, joiden sisältö ei muutu yhden web CMS -järjestelmän kutsun yhteydessä. Ohjelmakoodista saatettiin tällöin korvata näitä malliluokkien toistuvia objekteja luotujen globaalien objektien avulla.

Objektien käyttöä parannettiin web CMS -järjestelmässä myös lisäämällä luokkien alustamattomien objektien luontia ennen luuppeja, joiden sisällä oli aiemmin luotu aina uusia objekteja alustettavaksi. Näissä luupeissa saatettiin tämän muutoksen myötä vain kloonata alustamaton objekti uudeksi objektiksi, mikä poisti näiltä objekteilta tarpeen suorittaa luokkien rakentajissa tehtäviä toimenpiteitä jokaisen luupin kierroksen kohdalla.

6.3 Keskeisten funktioiden ja tietorakenteiden parantaminen

Web CMS -järjestelmän keskeisten funktioiden suorituskyvyn ongelmat koskevat luvussa 5 kuvatulla tavalla oikeastaan kaikkia järjestelmään kohdistuvia kutsuja. Samalla tapaa myös ongelmalliset tietorakenteet ovat sellaisia, jotka aiheuttavat ongelmia lähes kaikkien järjestelmään kohdistuvien kutsujen kohdalla. MVC-mallin mukainen raskas järjestelmän rakenne ja kehys laskevat väistämättä suorituskykyä [Wang 2011]. Tällöin on erittäin tärkeää, että keskeiset usein käytettävät funktiot on muotoiltu käyttäen suorituskyvyn kannalta parhaimpia käytäntöjä.

Web CMS -järjestelmän keskeisten funktioiden suorituskykyä pyrittiin parantamaan kahdella erilaisella teemalla. Ensin keskeisistä funktioista pyrittiin eriyttämään uusia funktioita Fowler *et al.* [1999] kuvaamin refaktoroinnin keinoin funktiokohtaisen laskentamäärän pienentämiseksi. Tämä pyrittiin kohdentamaan funktioissa sellaiseen käsittelyyn, mitä suoritetaan vain hyvin harvoin näiden funktioiden yhteydessä, mutta joiden ehtolausekkeet kuitenkin käsitellään hyvin usein funktiokutsujen yhteydessä. Esimerkiksi järjestelmän asetusten arvojen haussa haetaan asetuksen avaimella asetuksen arvoa eri asetusryhmistä. Eri asetusten ryhmiä voi olla web CMS -järjestelmässä käytössä kahdesta esimerkiksi neljään riippuen siitä, millaisia moduuleja, kuten tutkielmassa usein viitattu verkkokauppajärjestelmä, järjestelmässä on käytössä. Kaikkia näitä saatavilla olevia eri asetusten ryhmiä yritetään kuitenkin tarkastella haettaessa asetuksen arvoa erityisesti siinä tapauksessa, että asetus on sellainen, ettei sitä ole järjestelmässä. Tällaisia asetuksia ovat muun muassa asiakaskohtaiset asetukset, jolloin näitä funktiokutsuja suoritetaan hyvin paljon yhden järjestelmän kutsun aikana.

Web CMS -järjestelmän keskeisiä funktioita kutsutaan sivusta riippuen sadoista kerroista jopa kymmeniin tuhansiin kertoihin yksittäisen kutsun yhteydessä porautuvan mittariston tuottamien tietojen perusteella. Näin ollen keskeisten funktioiden

suorituskyvyn parantamisen toinen teema kohdistuikin siihen, että nämä funktiot olisi muotoiltu mahdollisimman suorituskykyisiksi. Funktioiden rakenteiden muokkauksessa hyödynnettiin aktiivisesti erilaisia Bradley [2018], Arsenault [2017] ja Mohammad [2017] verkkojulkaisujen perusteella suorituskykyisimmiksi todettuja PHP:n käytäntöjä.

Web CMS -järjestelmän vanhempien raskaiden tietorakenteiden suorituskyvyn perusongelmien ratkaisemiseksi olisi todennäköisesti tarvittu huomattavasti enemmän aikaa kuin tämän tutkielman parissa oli niihin mahdollista käyttää. Esimerkiksi sivupohjamoottorin osalta jo tämän luvun muissa alaluvuissa käsitelty välimuistin lisäys todettiin toistaiseksi riittävänä toimenpiteenä suorituskyvyn parantamiseksi ja optimoinniksi. Lisäksi sivupohjamoottorissa optimoitiin ja parannettiin joidenkin sen keskeisten funktioiden toimintaa.

Web CMS -järjestelmän vanhempien tietorakenteiden suorituskyvyn ongelmia pyrittiin kuitenkin myös ratkaisemaan purkamalla näistä pois niihin vähitellen pesiytyneitä arkkitehtuuria rappeuttavia ja ohjelmakoodin ymmärrettävyyttä laskevia erilaisia järjestelmän toiminnan kannalta jo vanhentuneita kohteita. Nämä kohteet ovat todennäköisesti syntyneet vähitellen tarpeesta laajentaa, soveltaa ja muokata näitä eri tietorakenteita [Neill ja Laplante 2006]. Esimerkiksi navigaatiohierarkian luonti sisälsi jonkin verran sellaista käsittelyä, mitä oli vain hyvin pitkälti lisätty jo olemassa olevan käsittelyn sisään ja päälle. Tällöin tämän järjestelmän osion suorituskyky laskee väistämättä vähitellen tarvittavan laskennan ja käsittelyn määrän kasvaessa.

Tutkielman viimeisessä luvussa 7 käsitellään tapaustutkimuksen kohteena olevan web CMS -järjestelmän suorituskyvyn optimointi- ja parannustoimenpiteiden tuomia suorituskyvyn muutoksia porautuvan ja yleismittariston perusteella. Tuloksia käsitellään yhteisesti kaikkien 12 kehitysiteraation tuottamien tulosten perusteella.

7 Tulokset

Tapaustutkimuksen kohteena olevaan web CMS -järjestelmään tehtiin yhteensä 12 kehitysiteraatiota, joiden aikana pyrittiin parantamaan ja optimoimaan sen suorituskykyä soveltaen olemassa olevia erilaisia ratkaisuja. Ratkaisujen toteuttamisen keskiössä oli myös järjestelmän teknisen velan määrän laskeminen ja kokonaisarkkitehtuurin tilan parantaminen esimerkiksi refaktoroinnin avulla. Web CMS -järjestelmän teknistä velkaa pyrittiin siis maksamaan pois tutkielman yhteydessä toteutettujen ratkaisujen avulla [Buschmann 2011].

Tässä luvussa vertaillaan suorituskyvyn alkutilanteen ja lopputilanteen eroja yleismittariston ja porautuvan mittariston tulosten perusteella. Lisäksi luvussa esitetään kriittinen näkökulma saavutettuihin tuloksiin sekä jatkokehitysehdotukset tapaustutkimuksen web CMS -järjestelmän suorituskyvyn ja laatutilanteen parantamiseksi myös tulevaisuudessa.

Tulosjoukko	Alkutilanne keskikoko kt	Lopputilanne keskikoko kt	Muutos %
Alatuoteryhmäsivujen keskiarvo	3940,5	1811,5	-54,0
Alatuoteryhmäsivujen tuotelistan keskiarvo	26559,5	11127	-58,1
Etusivu	36194	5547	-84,7
Tuotesivujen keskiarvo	31229,5	3154	-89,9
Variaatiotuotesivujen keskiarvo	44350	5484	-87,6
Sisältösivu 1	3520	1505	-57,2
Päätuoteryhmien sivu	3671	1679	-54,3

Taulukko 4. Web CMS -järjestelmän profilointitiedostojen koon muutos porautuvan mittariston perusteella.

Taulukossa 4 on eritelty web CMS -järjestelmän porautuvan mittariston testien tuottamien profilointitiedostojen koon muutosta alkutilanteen ja lopputilanteen välillä. Kuten tutkielman luvussa 5 todettiin, kuvaavat profilointitiedostojen koot järjestelmän kompleksisuutta. Taulukon 4 mukaisesti profilointitiedostot pienenivät vähintään 50 % jokaisessa tulosjoukossa, mutta monimutkaisista tulosjoukoista yli 80 % kaikilla paitsi tulosjoukolla ”Alatuoteryhmäsivujen tuotelistan keskiarvo”. Tämä on erittäin merkittävä ja positiivinen muutos suhteessa alkutilanteeseen ja kertoo järjestelmään kohdistuvien kutsujen aikana suoritettavan laskenta- ja käsittelymäärän kompleksisuuden merkittävästä laskusta. Koska ohjelmistot muuttuvat elinkaariensa aikana tyypillisesti kompleksisimmiksi kokonaisarkkitehtuurin rappeutumisen myötä [Neill ja Laplante 2006], voidaan myös todeta, että web CMS -järjestelmän kokonaisarkkitehtuuria pystyttiin parantamaan ja optimoimaan tehtyjen toimenpiteiden avulla.

Taulukon 4 perusteella voidaan myös todeta, että vaikka MVC-mallin mukaisen järjestelmän laajan kehyksen käyttö vaatii aina jonkin verran laskentaa ja käsittelyä järjestelmään kohdistuvien kutsujen yhteydessä [Wang 2011], on sen merkitystä kuitenkin onnistuttu pienentämään kaikkien sivujen kohdalla. Tämä korostuu erityisesti yksinkertaisten tulosjoukkojen kohdalla, sillä näissä järjestelmän kehyksen rooli on ollut laskennallisesti eniten käsittelyä ja suoritusta vaativa web CMS -järjestelmän osa testien yhteydessä.

Sivu	Alkutilanne RAM-allokaatio kt	Lopputilanne RAM-allokaatio kt	Muutos %
Alatuoteryhmä 101	6144	5957,8	-3,0
Alatuoteryhmä 30301	6144	6144	0
Frontpage	8396,8	6144	-26,8
Leikkipallo	8192	6144	-25
Lisätuote_100_3	8192	6144	-25
Paita	8192	6144	-25
Televisio	10240	8192	-20
Testisivu 1	6144	5957,8	-3,0
Tuotteet	6144	6702,5	9,1

Taulukko 5. Web CMS -järjestelmän RAM-allokaation muutos.

Nopean PHP-pohjaisen järjestelmän riskinä on käyttää enemmän muistia, jolloin järjestelmän kyky esimerkiksi skaalautua kysynnän kasvaessa heikkenee [Arsenault 2017]. Taulukosta 5 on kuitenkin nähtävissä, että web CMS -järjestelmän tapauksessa järjestelmän RAM-allokaatio ei kasvanut kuin yhden yksinkertaisen sivun yhteydessä: ”Tuotteet”-sivulla allokaatio kasvoi 9,1 %, kun se pysyi suurin piirtein samana tai laski hieman muiden yksinkertaisten sivujen yhteydessä.

Monimutkaisten sivujen yhteydessä RAM-allokaatio laski noin 25 % kaikkien paitsi ”Televisio”-sivun yhteydessä, jossa se laski kuitenkin 20 %. Välimuistin käytön lisääminen tutkielman luvussa 6 esitellyin tavoin lisäsi todennäköisesti RAM-allokaatiota jonkin verran kaikkien sivujen yhteydessä. Samalla web CMS -järjestelmän RAM-allokaatio kuitenkin laski toimenpiteiden toiston, objektien ja datan tuhlailevan käytön sekä muiden tehtyjen toimenpiteiden takia. Näin ollen lopputulemana järjestelmän RAM-allokaationkin osalta pystyttiin tuottamaan merkityksellinen ja positiivinen tulos tehtyjen toimenpiteiden avulla. RAM-allokaation muutoksen perusteella web CMS -järjestelmän pitäisi myös skaalautua jatkossa paremmin suuremmille käyttömäärille [Arsenault 2017].

Taulukossa 6 on eritelty web CMS -järjestelmän SQL-kyselyjen määrän muutosta. Yksinkertaisten sivujen tapauksessa SQL-kyselyjen määrä laski kaikilla sivuilla hieman yli 40 %. Monimutkaisten sivujen kohdalla saavutettu muutos oli

kuitenkin erittäin merkittävä erityisesti tulosjoukkoihin ”Etusivu” ja ”Tuotesivu” kuuluvien sivujen ”Frontpage”, ”Leikkipallo” ja ”Lisätuote_100_3” kohdalla; näillä sivuilla SQL-kyselyjen määrä laski lähes 95 %. Lisäksi monimutkaisten sivujen kohdalla ”Variaatiotuotesivu”-tulosjoukkoon kuuluvien ”Paita”- ja ”Televisio”-sivujen SQL-kyselyjen määrät laskivat noin 77 % ja 68 %.

Sivu	Alkutilanne SQL-kyselyjen lkm	Lopputilanne SQL-kyselyjen lkm	Muutos %
Alatuoteryhmä 101	110,1	60,7	-44,9
Alatuoteryhmä 30301	111	63,3	-43,0
Frontpage	2942,4	153,7	-94,8
Leikkipallo	2073,3	114,5	-94,5
Lisätuote_100_3	2074	114,3	-94,5
Paita	2750	626,3	-77,2
Televisio	3723,1	1178,3	-68,4
Testisivu 1	87,1	45	-48,3
Tuotteet	99,2	56,3	-43,2

Taulukko 6. Web CMS -järjestelmän SQL-kyselyjen muutos.

Kuten Sa´adah *et al.* [2015] totesi, osoittautui välimuistin käytön laajentaminen siihen soveltuvissa tietokantakutsuissa erittäin tehokkaaksi keinoksi vähentää web CMS -järjestelmän kutsujen yhteydessä suoritettavien SQL-kyselyjen määrää. Tietokantakyselyjen määrä pieneni myös luvussa 6 esiteltyjen toiston, objektien ja datan tuhlailevan käytön vähentämisen myötä. Tämän merkitys on myös keskeinen web CMS -järjestelmään kohdistuvien ensimmäisten kutsujen kohdalla, jolloin näiden kutsujen kohdalla suoritettavien SQL-kyselyjen tuloksia ei ole vielä tallennettuna välimuistiin.

Tämän tutkielman lähtöasetelmana oli saavuttaa tutkimuskysymyksiin vastaamisen kautta sellaisia tuloksia, jotka laskisivat tapaustutkimuksen kohteena olevaan web CMS -järjestelmään kohdistuvien kutsujen suoritusaikoja, sillä pitkät suoritusajat olivat muodostuneet järjestelmässä ongelmallisiksi. Taulukoiden 4, 5 ja 6 mukaisesti järjestelmän suorituskyky onkin parantunut jo merkittävästi eri osa-alueilla toteutettujen toimenpiteiden myötä.

Taulukossa 7 on eritelty web CMS -järjestelmän sivujen suoritusaikojen muutosta. Suoritusajan pienennys yksinkertaisten sivujen tapauksessa on ollut 30 % molemmin puolin muiden paitsi ”Tuotteet”-sivun kohdalla. Tälläkin sivulla suoritusaika laski kuitenkin noin 10 %. Monimutkaisten sivujen tapauksessa suoritusajan lasku on erittäin merkittävä; kaikkien monimutkaisten sivujen tapauksessa lasku on ollut vähintään 69 %. SQL-kyselyjen lukumäärän muutoksen mukaisesti myös suoritusaikojen lasku tulosjoukkoihin ”Etusivu” ja ”Tuotesivu” kuuluvien sivujen tapauksessa on peräti yli 90 %.

Sivu	Alkutilanne suoritusaika s	Lopputilanne suoritusaika s	Muutos %
Alatuoteryhmä 101	0,205430007	0,129737832	-36,8
Alatuoteryhmä 30301	0,298458767	0,221150268	-25,9
Frontpage	3,358900571	0,322730491	-90,4
Leikkipallo	2,574868417	0,241635366	-90,6
Lisätuote_100_3	2,58102138	0,222141526	-91,4
Paita	3,082623553	0,701790419	-77,2
Televisio	4,494285297	1,377685612	-69,3
Testisivu 1	0,181891528	0,11748004	-35,4
Tuotteet	0,184144519	0,166130651	-9,8

Taulukko 7. Web CMS -järjestelmän suoritusajan muutos.

Web CMS -järjestelmän suorituskyvyn parannus- ja optimointitoimenpiteiden avulla saavutettu suoritusajan lasku on erittäin merkittävä kaikkien testauksen kohteena olleiden sivujen kohdalla. Suoritusajan lasku on ollut erittäin suuri erityisesti monimutkaisten sivujen kohdalla, jotka olivatkin web CMS -järjestelmän kaikista ongelmallisimpia kohteita. Mittaristojen avulla testatuista sivuista ainoa sivu, jonka suoritusaika on yleismittariston avulla suoritettujen testien perusteella vielä yli yhden sekunnin, on ”Variaatiotuote”-tulosjoukon sivu ”Televisio”, jossa parannus oli kuitenkin myös erittäin merkittävä.

Vaikka tapaustutkimuksen kohteena olevan web CMS -järjestelmän suorituskykyä saatiinkin parannettua porautuvan mittariston ja yleismittariston perusteella, on hyvä kiinnittää huomiota myös käytettyjen mittaristojen ongelmakohtiin. Mittaamiselle on tyypillistä, että se saa ihmiset keskittymään vain niihin asioihin, joiden avulla saavutetaan hyviä tuloksia näissä mittareissa [Laird ja Brennan 2006]. Esimerkiksi tässä tutkielmassa web CMS -järjestelmän tapauksessa käytetyt testitapaukset eivät kata kaikkia erilaisia käyttötapauksia ja tilanteita tai sivutyyppejä järjestelmässä. Tällöin suorituskyvyn todellinen taso voi myös olla hyvin erilainen jollekin asiakasyrityksen asiakkaalle asennetussa web CMS -järjestelmässä, jossa on otettu käyttöön esimerkiksi hyvin eroavia toimintoja. Lisäksi mittaristojen käyttö vaatii web CMS -järjestelmän tyypillisestä webhotelli-pohjaisesta asennusympäristöstä johtuen jonkin verran manuaalista työtä, mikä nostaa todennäköisesti mittaristojen tuottaman datan virhemarginaalia jonkin verran. Virhemarginaalia nostaa myös yleismittariston avulla suoritettujen yksittäisten sivujen testien lukumäärä yhdellä testauskierroksella, mikä on kymmenen. Tällöin mittaristojen tuottama data ei ole välttämättä täysin todellisuutta heijastavaa ja tarkkuudeltaan täysin optimaalista [Laird ja Brennan 2006].

Annetusta kritiikistä huolimatta web CMS -järjestelmän suorituskykyä pystytään siis parantamaan tehokkaasti tässä tutkielmassa esiteltyjen keinojen avulla tässä luvussa esiteltyjen tuloksien perusteella. Keskeisiksi toimenpiteiksi web CMS -järjestelmän suorituskyvyn parantamisessa osoittautuivat mittaristojen tuottamien tietojen perusteella seuraavat keinot: järjestelmän kehyksen ja keskeisten funktioiden parantaminen tehokkaammiksi, toimintojen kapselointi erilaisten käsittelyjen keskittämiseksi, datan tuhlailevan käytön ja toiston purkaminen sekä objektien käytön optimointi ja välimuistin käytön lisääminen eri puolilla järjestelmää. Myös useissa eri lähteissä, kuten Fowler *et al.* [1999], Tsantalis ja Chatzigeorgiou [2010], Fokaeds *et al.* [2012], Rocha *et al.* [2017], kuvatuilla erilaisilla refaktoroinnin keinoilla oli keskeinen vaikutus web CMS -järjestelmän suorituskyvyn parantamisessa ja samalla myös järjestelmän kokonaisarkkitehtuurin parantamisessa sekä teknisen velan vähentämisessä.

Tämän tutkielman yhteydessä tehdyt kehitystoimenpiteet on sisällytetty tapaustutkimuksen kohteena olevaan web CMS -järjestelmään, ja ne ovat jatkossa myös saatavilla sen uusissa versioissa.

7.1 Jatkokehitysehdotukset

Kuten Silva ja Balasubramaniam [2012] totesivat, nopea järjestelmä voi olla arkkitehtuuriltaan erittäin rappeutunut ja hajota pienimmänkin muutoksen myötä. Vaikka tapaustutkimuksen kohteena olevan web CMS -järjestelmän suorituskykyä pystyttiin parantamaan ja optimoimaan tämän tutkielman yhteydessä merkittävästi, sisältää se edelleen runsaasti teknistä velkaa ja sen kokonaisarkkitehtuuri on kohtalaisen rappeutunut. Tämä ilmenee muun muassa hankalasti ymmärrettävinä rakenteina ja suorituspolkuina, vaikeasti muokattavissa olevina toimintoina, erilaisten virheiden ja pienten muutosten korkeana määränä, pikakorjauksina ja arkkitehtuurin tasolta suuresti eroavina järjestelmän osina [Neill ja Laplante 2006, Eick *et al.* 2001]. Kokonaisarkkitehtuuri on rappeutunut ja tekninen velka kerääntynyt todennäköisesti web CMS -järjestelmään vähitellen vuosien kuluessa olosuhteiden seurauksena.

Järjestelmän parissa työskentelevät henkilöt ovat vaihtuneet, vaatimuksien määritelmät ovat olleet vaihtelevia, aikataulupaineet kovia ja laadunhallinta ylipäätään huonolla tasolla; nämä ovat erittäin tyypillisiä ja yleisesti tunnistettuja syitä kokonaisarkkitehtuurin rappeutumiselle ja kaikkinensa teknisen velan kertymiselle [Eick *et al.* 2001, Buschmann 2011, Tunttunen 2014, Rocha *et al.* 2017]. Tulevaisuudessa onkin erittäin tärkeää, että järjestelmän laadullista tilaa priorisoidaan uusien ominaisuuksien yli teknisen velan määrän laskemiseksi [Tunttunen 2014]. Tällöin ohjelmiston kehitys- ja ylläpitokulut myös tyypillisesti laskevat [Buschmann 2011, Rocha *et al.* 2017].

Ohjelmiston kokonaisarkkitehtuurin rappeutumista voidaan hallita esimerkiksi tarkkailemalla koko järjestelmän evoluutiota kokonaisuutena versionhallinnan avulla [Silva ja Balasubramaniam 2012]. Tapaustutkimuksen web CMS -järjestelmässä

laadunhallinnan prosessi hakee vielä tämän tutkielman kirjoitushetkellä muotoansa ja esimerkiksi koodikatselmointien käyttö osana järjestelmän versionhallintaa on noussut keskeiseen osaan vasta viime aikoina. Koodikatselmointien käyttö onkin erittäin hyväksi havaittu keino teknisen velan syntymisen estämiseksi [Rocha *et al.* 2017], joten tämä on askel oikeaan suuntaan. Lisäksi jatkuva ohjelmakoodin refaktorointi pienissä erissä on tärkeä keino arkkitehtuurin rappeutumisen hidastamiseksi ja estämiseksi [Fowler *et al.* 1999]. Kaikkia teknisen velan pienentämiseen liittyviä toimenpiteitä tulisi myös arvioida taloudellisesta näkökulmasta, jotta pystytään valitsemaan kuhunkin tilanteeseen liiketoiminnan näkökulmasta paras lähestymistapa [Buschmann 2011].

Tärkeintä laadun hallinnassa on kuitenkin jatkuva laatutason seuranta ja se, että laadukkaan järjestelmän tunnuspiirteet, tavoitetasot ja määritelmät sekä noudatettava laadunhallinnan prosessi on dokumentoitu ja että dokumentaatio on kaikkien järjestelmän parissa toimivien tahojen käytettävissä riippumatta heidän roolistaan järjestelmän parissa [Sommerville 2007]. Selkeän laatukäsikirjan tuottaminen käytettäväksi tapaustutkimuksen web CMS -järjestelmän tuotekehityksessä olisikin järkevä askel laadun parantamisessa. Jotta web CMS -järjestelmän laatutasoa pystyttäisiin nostamaan tehtyjen määritelmien avulla, tulisi jokaisen järjestelmän parissa työskentelevän noudattaa määriteltävää prosessia aina. Muutoin riskinä on laatutason heikkeneminen lyhyen tähtäimen hyötyjen saavuttamiseksi [Rocha *et al.* 2017].

Web-pohjaiset järjestelmät ovat tyypillisesti maineeltaan huonosti rakennettuja eikä PHP kannusta myöskään hyvien käytäntöjen vaalimiseen ohjelmistokehityksessä [Nederlof *et al.* 2014, Cholakov 2008]. Jotta kehittäjien mahdollinen vaihtuvuus, eri kehittäjien välinen järjestelmän tuntemuksen taso tai kehittäjien eri taitotasot eivät vaikuttaisi ainakaan niin suuresti tuotettuihin ratkaisuihin ja niiden laatutasoon, olisi laatukäsikirjan hyvä sisältää tarkat määritelmät erilaisista järjestelmän arkkitehtuuria koskevista säännöistä, rakenteista ja ohjelmakoodin formatoinnista. Tällöin näihin voidaan aina viitata, kun järjestelmään tulevia muutoksia evaluoidaan koodikatselmointien yhteydessä [Silva ja Balasubramaniam 2012]. Ihmisille on tyypillistä, että asiat ymmärretään eri tavalla ja määritellään hieman eri tavalla [Schneidewind 1996]. Sisäinen laatukäsikirja voisi toimia organisaation sisäisenä laatustandardina, jolloin se parantaisi laatua lisäämällä ymmärryksen tasoa eri tahojen välillä [Schneidewind 1996].

Kun web CMS -järjestelmän laatutaso nousee ja erilaisten virheiden esiintyvyyden määrä laskee, on todennäköistä, että myös järjestelmän suorituskyvyn taso jatkaa parantumistaan [Laird ja Brennan 2006]. Myös edellytykset asiakasyrityksen asiakkaiden sivustojen ja niihin liitettyjen verkkokauppojen menestykseen ovat tällöin paremmat [Chosin ja Ghaffari 2017], kun web CMS -järjestelmän laatutaso paranee ja suorituskyky kasvaa, jolloin koko infrastruktuuri paranee.

8 Yhteenveto

Tämän tutkielman tavoitteena oli löytää keinoja, joilla web CMS -järjestelmien suorituskykyä voidaan optimoida ja parantaa. Toisena tavoitteena oli toteuttaa näitä löydettyjä keinoja tapaustutkimuksen kohteena olleeseen web CMS -järjestelmään sen määrittämiseksi, pystytäänkö kyseisen järjestelmän suorituskykyä optimoimaan ja parantamaan löydetyillä keinoilla ja ratkaisuilla.

Tutkielmassa tehtiin kirjallisuuskatsaus CMS-järjestelmiin ja ohjelmistojen laatuun, laadun arviointiin ja erilaisiin metriikoihin. CMS-järjestelmien osalta käsiteltiin erityisesti web CMS -järjestelmiä ja niissä dynaamisia web CMS -järjestelmiä sekä niiden tyypillisiä ominaispiirteitä. Laadun osalta käsiteltiin myös laatua osana ohjelmistojen elinkaarta ja miten laatua voidaan parantaa muun muassa refaktoroinnin ja koodikatselmointien avulla. Lisäksi käsiteltiin lyhyesti erilaisia laatujärjestelmiä ja standardeja.

Tapaustutkimuksen kohteena olevaan web CMS -järjestelmään määriteltiin tutkielman yhteydessä vakioitu testiympäristö sekä järjestelmän testauskonfiguraatio. Lisäksi järjestelmän testidataa laajennettiin merkittävästi kattamaan yleisimpiä tarvittavia tapauksia. Tutkielman yhteydessä luotiin järjestelmään myös kaksi erilaista suorituskyvyn mittaristoa: yleismittaristo ja porautuva mittaristo. Yleismittaristo tallentaa yleisiä järjestelmän backend-puolen suoritusarvoja .csv-tiedostoon: yksittäisten kutsujen suoritusajan, SQL-kyselyjen lukumäärän ja RAM-allokaation. Porautuva mittaristo perustuu PHP:n Xdebug-laajennoksen avulla suoritettavaan ohjelmakoodin profilointiin, joka suoritetaan rajatusti kutsujen yhteydessä.

Yleismittariston ja porautuvan mittariston avulla suoritettujen testien perusteella kävi ilmi, että tapaustutkimuksen kohteena olevan web CMS -järjestelmän suorituskyvyn ongelmat johtuivat seuraavanlaisista asioista: tietokantakyselyjen erittäin suuri määrä, kehykselle keskeisten funktioiden hitaus, datan ja objektien tuhlaileva käyttö, vanhojen tietorakenteiden hitaus, turha toisto ja ylipäätään kokonaisarkkitehtuurin rappeutuminen. Nämä ongelmat kielivät web CMS -järjestelmässä piilevästä teknisestä velasta ja kokonaisarkkitehtuurin rappeutumisesta.

Web CMS -järjestelmän suorituskykyä parannettiin toteuttamalla toimenpiteitä, joilla ongelmallisiksi osoittautuneita järjestelmän osioita optimoitiin ja parannettiin. Olennaisena osana valituissa ratkaisuissa oli myös web CMS -järjestelmän kokonaisarkkitehtuurin tilan ja kehityksen ottaminen huomioon. Ratkaisut pyrittiin löytämään soveltamalla kirjallisuudesta löydettyjä erilaisia keinoja kohdejärjestelmässä. Ratkaisuiksi muodostuivat seuraavat keinot: tietokantakyselyjen määrän vähentäminen, välimuistin käytön lisääminen, objektien käytön tehostaminen, keskeisten funktioiden ja tietorakenteiden parantaminen, käsittelyn keskittäminen ja toiston purkaminen sekä toiminnallisuuksien kapselointi ja abstraktointi.

Suoritetuilla toimenpiteillä saavutettiin merkittäviä suorituskyvyn parannuksia tapaustutkimuksen web CMS -järjestelmässä sekä yleismittariston että porautuvan mittariston tulosten perusteella: suorituskyky parani merkittävästi kaikilla mittaristojen metriikoilla. Tämän tapaustutkimuksen web CMS -järjestelmän suorituskykyä pystytään siis parantamaan merkittävästi tässä tutkielmassa esitellyillä eri keinoilla, vaikka kuhunkin järjestelmään soveltuvat keinot ovat aina tapauskohtaisia.

Jatkon kannalta tapaustutkimuksen web CMS -järjestelmän tuotekehityksessä suositetaan tutkielman perusteella otettavaksi laadunhallinta kokonaisuudessaan paremmin huomioon, jotta järjestelmään kertyneen teknisen velan määrää voidaan vähentää ja kokonaisarkkitehtuurin tasoa parantaa - tämä on keskeistä myös tuotekehityskulujen laskemisessa ja suorituskyvyn parantamisessa jatkossa.

Viiteluettelo

[Amarasinghe 2016] Sean Amarasinghe. 2016. *Service Worker Development Cookbook*. Packt Publishing Ltd.

[Alcocer ja Bergel 2015] Juan Pablo Sandoval Alcocer, Alexandre Bergel. 2015. Tracking down performance variation against source code evolution. In: *Proceedings of the 11th Symposium on Dynamic Languages*, 129-139

[Alieksieiev ja Bondarenko 2012] Mykola Alieksieiev, Volodymyr Bondarenko. 2012. Analysis of approaches used to raise productivity of PHP programs. In: *Proceedings of International Conference on Modern Problem of Radio Engineering, Telecommunications and Computer Science*, 349.

[Arsenault 2017] Cody Arsenault. 2017. Improving php performance for web applications. In: *www.keycdn.com*, *https://www.keycdn.com/blog/php-performance*. Published: 23 Mar 2017. Accessed on: 2.1.2019

[Austin 2001] Robert D. Austin. 2001. The effects of time pressure on quality in software development: An agency model. *Information Systems Research*, 12, 2, Jun. 2001, 195-207

[Barker 2016] Deane Barker. 2016. *Web Content Management, Systems, Features, and Best Practices.* O'Reilly Media.

[Boiko 2001] Bob Boiko. 2001. Understanding content management. *Bulletin of the American society for information science and technology*, 8-13

[Boiko 2005] Bob Boiko. 2005. *Content Management Bible, 2nd Edition*. Wiley Publishing, Inc.

[Bradley 2018] Steven Bradley. 2018. Tips for writing better performing php code. In: *vanseodesign.com*, *https://vanseodesign.com/web-design/performance-tips-php/*. Published: 26 Jun 2018. Accessed on: 2.1.2019

[Bøegh 2008] Jørgen Bøegh. 2008. A new standard for quality requirements. *IEEE Software*, 25, 2, Mar-Apr 2008, 57-63

[Buschmann 2011] Frank Buschmann. 2011. To pay or not pay technical debt. *IEEE Software,* 28, 6, 29-31.

[Chaniotis *et al.* 2015] Ioannis K. Chaniotis, Kyriakos-Ioannis D. Kyriakou, Nikolaos D. Tselikas. 2015. Is Node.js a viable option for building modern web applications? A performance evaluation study. *Computing*, Oct. 2015, 97, 10, 1023-1044.

[Cholakov 2008] Nikolaj Cholakov. 2008. On some drawbacks of the PHP platform. In: *Proceeding CompSysTech '08 Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, article 12

[Chosin ja Ghaffari 2017] Mahdi Chosin, Ali Ghaffari. 2017. An investigation of the impact of effective factors on the success of e-commerce in small- medium-sized companies. In: *Computers in Human Behavior,* 66, 67-74.

[Cu *et al.* 2009] Wei Cu, Lin Huang, LiJing Liang, Jing Li. 2009. The research of PHP development framework based on MVC pattern. In: *IEEE 2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, 947-949

[Eick *et al.* 2001] Stephen G. Eick, Todd L. Graves, Alan F. Karr, J.S. Marron, Audris Mockus. 2001. Does code decay? Assessing the evidence from change management data. *IEEE Transactions on Software Engineering,* 27, 1, Jan. 2001

[Eisenhardt 1989] Kathleen M. Eisenhardt. 1989. Building theories from case study research. *Academy of Management Review*, 14, 4, 532-550.

[Fenton 1996] Norman Fenton. 1996. *Improve Quality?* Centre for Software Reliability, City University, London.

[Fokaeds *et al.* 2012] Marios Fokaeds, Nikolaos Tsantalis, Eleni Stroulia, Alexander Chatzigeorgiou. 2012. Identification and application of extract class refactorings in object-oriented systems. *The Journal of Systems and Software*, Oct. 2012, 85, 10, 2241-2260.

[Fowler *et al.* 1999] Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts. 1999. *Refactoring: Improving the Design of Existing Code.*

[Gavalda 2019] Mark Gavalda. 2019. The definitive php 5.6, 7.0, 7.1, 7.2 & 7.3 benchmarks (2019). In: *kinsta.com*, *https://kinsta.com/blog/php-benchmarks/*. Updated: 14 Jan 2019, Accessed on: 10.3.2019

[Goldman ja Narayanaswamy 1992] Neil Goldman, K. Narayanaswamy. 1992. Software evolution through iterative prototyping. In: *International Conference on Software Engineering*, 158-172, May 1992

[Griffith *et al.* 2011] Isaac Griffith, Scott Wahl, Clemente Izurieta. 2011. Evolution of legacy system comprehensibility through automated refactoring. In: *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, 35-42.

[Grubor ja Jakša 2018] Aleksandar Grubor, Olja Jakša. 2018. Internet marketing as a business necessity. Interdisciplinary Description of Complex Systems, 16, 2, 265-274.

[Hassaine *et al.* 2012] Salima Hassaine, Yann-Gaël Guéhéneuc, Sylvie Hamel, Giuliano Antoniol. 2012. ADvISE: Architectural decay in software evolution. In: *16th European Conference on Software Maintenance and Reengineering.* 267-276

[Hauzar ja Kofroň 2014] David Hauzar and Jan Kofroň. 2014. WeVerca: Web Applications verification for PHP. In: *Proceedings of 12th International Conference on Software Engineering and Formal Methods*, 296-301

[Hevner et al. 2004] Alan R. Hevner, Salvatore T. March, Jinsoo Park. 2004. Design science in information systems research. *MIS Quarterly*, 28, 1, 57-105, Mar 2004.

[Hills *et al.* 2014] Mark Hills, Paul Klint and Jurgen J. Vinju. 2014. Static, lightweight includes resolution for PHP. In: *Proceedings of the 29th International Conference on Automated Software Engineering,* 503-513

[Jose-Manuel et al. 2018] Martinez-Caro Jose-Manuel, Antonio-Jose Aledo-Hernandez, Guillen Perez, Antonio Guillen-Perez, Ramon Sanchez-Iborra, Maria-Dolores Cano. 2018. A comparative study of web content management systems. Information, 9, 27, 15 pages.

[Kaimer ja Brune 2018] Fabian Kaimer, Philipp Brune. 2018. Return of the JS: Towards a Node.js-based software architecture for combined CMS/CRM applications. *Procedia Computer Science*, 141, 454-459.

[Komara *et al.* 2016] Hendra Komara, Bayu Hendradjaya, G.A. Putri Saptawati. 2016, Dynamic generic web pattern for multi-platform. In: *2016 International Conference on Data and Software Engineering*, 1-5.

[Laguna ja Crespo 2013] Miguel A. Laguna and Yania Crespo. 2013. A systematic mapping study on product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming*, 78, 8, 1010-1034.

[Laird ja Brennan 2006] Linda M Laird, M. Carol Brennan. 2006. *Software Measurement and Estimation.* John Wiley & Sons, Inc.

[Lyytinen ja Robey 1999] Kalle Lyytinen, Daniel Robey. 1999, Learning failure in information systems development. *Information Systems Journal*, 9, 85-101.

[Mens ja Tourwé 2004] Tom Mens, Tom Tourwé. 2004. A survey of software refactoring. In: *IEEE Transactions on Software Engineering*, Feb. 2004, 30, 2, 126-139

[Mohammad 2017] Atif Mohammad. 2017. An extensive checklist of php performance optimization techniques. In: *themoon.com*, https://theemon.com/extensive-checklist-php-performance-optimization-techniques/. Published: 25 Oct 2017. Accessed on: 2.1.2019

[Nederlof et al. 2014] Alex Nederlof, Ali Mesbah and Arie van Deursen. 2014. Software engineering for the web: The state of the practice. In: Proceeding ICSE Companion 2014 Companion Proceedings of the 36th International Conference on Software Engineering, 4-13

[Neill ja Laplante 2006] Colin J. Neill, Philip A Laplante. 2006. Paying down design debt with strategic refactoring. Computer, 39, 12, 113-116

[Ogunrinde ja Yoosuf 2016] Mutiat A. Ogunrinde, Tolani H. Yoosuf. 2016. Performance analysis on content management systems: a case study of Drupal and Joomla. *International Journal of Computational Engineering & Management*, 19, 2, 5-11

[PHP Sessions 2019] Php Session handling. In: https://www.php.net/manual/en/book.session.php Accessed on: 4.6.2019

[Pinpoint 2019] Pinpoint booking system for wordpress plugin. In: https://pinpoint.world/wordpress-booking Accessed on: 4.6.2019

[Popov *et al.* 2017] Nikita Popov, Biagio Cosenza, Ben Juurlink ja Dmitry Stogov. 2017. Static optimization in PHP 7. In: *Proceedings of the 26th International Conference on Compiler Construction*, 65-75.

[Rocha *et al.* 2017] Junior Cesar Rocha, Vanius Zapalowski and Ingrid Nunes. 2017. Understanding technical debt at the code level from the perspective of software developers. In: *SBES'17 Proceedings of the 31st Brazilian Symposium on Software Engineering*, 64-73

[Ruohonen *et al.* 2016] Jukka Ruohonen, Sami Hyrynsalmi, Ville Leppänen. 2016. Exploring the use of deprecated PHP releases in the wild internet: Still a LAMP issue? In: *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*, 1-12

[Sa'adah et al. 2015] Umi Sa'adah, Jauari Akhmad NH and Masfu Hisyam. 2015. Implementing singleton method in design of MVC-based PHP framework. In: Proceedings – 2015 International Electronics Symposium: Emerging Technology in Electronic and Information, 212-217

[Santos ja Gerosa 2018] Rodrigo Magalhães dos Santos, Macro Aurélio Gerosa. 2018. Impacts of coding practices on readability. In*: Proceedings of the 26th Conference on Program Comprehension*, 277-285

[Schneidewind 1996] Norman F. Schneidewind. 1996. *Do standards.* Information sciences, Naval Postgraduate School

[Silva ja Balasubramaniam 2012] Lakshitha de Silva, Dharini Balasubramaniam. 2012. Controlling software architecture erosion: A survey. *The Journal of Systems and Software,* 85, 132-151

[Sommerville 2007] Ian Sommerville. 2007. *Software Engineering Eight Edition*. Addison-Wesley

[Sotirovski 2001] Drasko Sotirovski. 2001. Heuristics for iterative software development. In: *IEEE Software*, 18, 3, 66-73.

[Tsantalis ja Chatzigeorgiou 2010] Nikolaos Tsantalis, Alexander Chatzigeorgiou. 2010. Identification of refactoring opportunities introducing polymorphism. *The Journal of Systems and Software*, Mar. 2010, 83, 3, 391-404.

[Tunttunen 2014] Panu Tunttunen. 2014. *Fighting technical debt: enabling sustainable productivity*. Pro gradu -tutkielma. Informaatiotieteiden yksikkö, Tampereen yliopisto.

[Vaidya et al. 2013] S.P. Vaidya, Vinod J. Kadam, Swpnil S.Dabhade, Pooja V. Dange, R. B. Gofankar. 2013. How to choose a website content management system. *National Conference On Emerging Trends In Academic Library.*

[Wang 2011] Guanhua Wang. 2011. Application of lightweight MVC-like structure in PHP. In: *2011 International Conference on Business Management and Electronic Information*, 74-77.

[Wiegers ja Beatty 2013] Karl Wiegers, Joy Beatty. 2013. *Software Requirements, Third Edition*. Mircosoft

[Woodside 2010] Arch G. Woodside. 2010. *Case study research: theory, methods, practice*. Emerald.

[Xdebug 2019] Derick Rethans. 2019. Xdebug extensions for php. In: xdebug.org, https://xdebug.org/ Accessed on: 25.4.2019

[Ying ja Miller 2012] Ming Ying and James Miller. 2012. Refactoring Legacy AJAX applications to improve efficiency of the data exchange component. *Journal of Systems and Software,* Jan. 2013, 86, 1, 72-88.